UNIVERSITY
of
GLASGOW

Enhancing face recognition by image warping

Jorge Garcia Bueno

Master Thesis in Electronics and Electrical Engineering No.77/2009

1st supervisor: **Dr. B. Porr**

2nd supervisor: **Prof. A.C. Bryce**

University of Glasgow, Scotland, 2009

Department of Electronics and Electrical Engineering

October 2009

"People are not remembered by the number of times they fail but for the number of times they succeed"

"Las personas no son recordadas por el número de veces que fracasan sino por el número de veces que tienen éxito"

Thomas Alba **Edison**, 1927.

UNIVERSITY OF GLASGOW

Abstract

University of Glasgow
Department of Electronics and Electrical Engineering

Bachelor of Engineering

by Jorge Garcia Bueno

This project has been developed as an improvement which could be added to the actual computer vision algorithms. It is based on the original idea proposed and published by *Rob Jenkins* and *Mike Burton* about the power of the face averages in artificial recognition.

The present project aims to create a new automated procedure applied for face recognition working with average images. Up to now, this algorithm has been used manually. With this study, the averaging and warping process will be done by a computer automatically saving large amounts of time. Through a clear user interface, the program that has been developed will receive a batch of face images of a person and will create an average picture of them deforming each one of them based on . Some settings (colours, size, etcetera ...) might be edited before any average is created and some options will be offered after the job is done to facilitate the addition of them to a face database. It is demonstrated in previous studies that the average picture generated contains most of the information of the group of original faces and therefore, a system would recognise this person easily than with any single image.

After the development of the software, a computational study will be done to locate the quality in terms of accuracy and speed of this solution. The program will be asked to learn a batch of faces of a group of people and afterwards it will be tested and compared with actual works to demonstrate if the algorithm is at the same level of quality.

Contents

List of Figures

List of Tables

Abbreviations

RGB	**R**ed **G**reen **B**lue
HSI	**H**ue **S**aturation **I**ntensity
HSV	**H**ue **S**aturation **V**alue
CCTV	**C**lose **C**ircuit **T**ele**V**ision
CPU	**C**ontrol **P**rocess **U**nit
PCA	**P**rincipal **C**omponent **A**nalisys
LDA	**L**ineal **D**iscriminant **A**nalisys
EBMG	**E**lastic **B**unch **G**raph **M**atching
CSA	**C**ross **S**egment **A**lgorithm
GPL	**G**eneral **P**ublic **L**icense
OCR	**O**ptical **C**aracter **R**ecognition

Dedicated to Rosalía, Jorge, Luis and Raquel
who recognise my face without problem . . .

Dedicado a Rosalía, Jorge, Luis y Raquel
que reconocen mi cara sin problemas . . .

Chapter 1

Resumen

1.1 Acta

El presente proyecto fue entregado y defendido como *Master Thesis* en el Departamento de Electrónica y Electricidad de la Universidad de Glasgow el día 25 de Mayo en el año 2009. Este trabajo ha sido tutelado diariamente por **Dr Bernd Porr** y seguido puntualmente por **Prof Catrina Bryce**. Como parte del jurado se incluyó a **Prof J O'Reilly** junto con los dos tutores anteriores.
La calificación obtenida en el proyecto fue **A2 (9.8)**.

Al mismo tiempo, en Leganés, **Dr Luis Moreno Lorente** siguió el trabajo de cerca como cotutor para desarrollar una investigación útil y aplicable. Por último, **Dr Julio Usaola** fue el responsable de que los trámites fueran satisfactorios trabajando como Coordinador Académico del convenio.

1.2 Presentación

El presente Proyecto de Fin de Carrera tiene como objetivo primordial desarrollar una aplicación capaz de mejorar los indices de reconocimiento facial actuales mediante un nuevo método basado en la generación de una cara media a partir de la suma de varias caras individuales de una persona. La cara media generada servirá como referencia de dicho sujeto. Según los estudios realizados por **Mike Burton** y **Rob Jenkins** en los últimos años [1], las probabilidades de reconocer a una persona basándose en su cara

[1]Profesores titulares del Departamento de Psicología en la Universidad de Glasgow

media aumentan exponencialmente respecto a un reconocimiento en base a una cara individual. Según estos estudios, el indice de reconocimiento puede a alcanzar el 100% de aciertos.

Con este Proyecto se demuestra que dicho proceso de *"averaging"* es correcto, aplicable y robusto mediante la implementación de un programa multiplataforma escrito en C++ usando las librerias Qt4 en el diseño del entorno gráfico. Los pasos seguidos en este texto para conseguir dicho objetivo han sido los siguientes:

1. **Conceptos básicos sobre el tratamiento digital de imágenes.** Resumen de los conceptos básicos repasando los algoritmos de deformación y pre/post procesamiento de imágenes digitales. También se plantea el problema y los pasos lógicos para llegar a una solución factible.

2. **Algoritmos de deformación.** Para poder realizar una media de las caras, es necesario deformar todas las caras individuales a una posición común. De esta forma todas las caracterísicas intrínsecas a cada imágen individual se encuentren en la misma posición antes de ser sumadas (*i.e.* ojos, nariz, boca , barbilla ...). En este proyecto se repasan los casos más generales de deformación de mallas. Entre ellos la transformación entre un par de líneas, transformación entre multiples líneas, deformación triangular mediante coordenadas baricéntricas (mostrado un ejemplo en la Figura 1.1) y finalmente deformación por mallas.

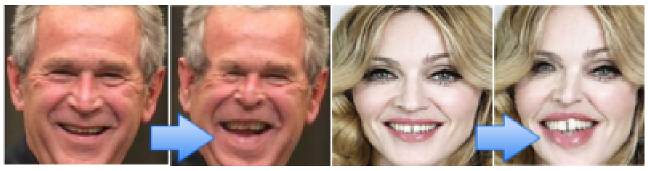


Figure 1.1: Resultados aplicando la deformación basada en mallas triangulares

3. **Selección de la malla.** Dependiendo de la geometría, forma y pose del objeto sobre el que se desea realizar la media, será necesario establecer una geometría y forma para la creación de la malla de puntos. En este paso se discuten varias opciones buscando aquella opción que optimiza el índice de reconocimiento facial. En la Figura 1.2 se muestran las dos mallas utizadas en este trabajo. La malla de la derecha corresponde al caso automático (pocos nodos, el método es impresiso pero muy rápido) mientras que la malla de la izquierda corresponde al caso manual (contiene más puntos haciéndo el proceso lento pero obteniendo resultados más precisos y robustos).

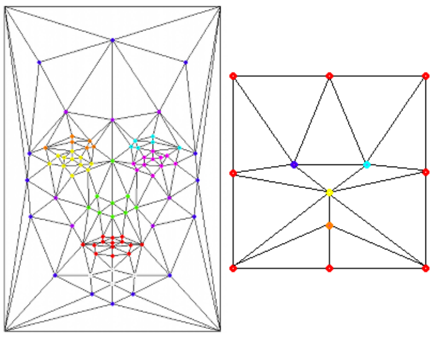


FIGURE 1.2: Mallas triangulares utilizadas para el análisis automático de patrones. Izquierda: malla de 34-nodos. Derecha: malla de 12-nodos.

4. **Automatización de la malla.** Para hacer el trabajo de selección de nodos de la malla, se implementó un método de reconociento de patrones basado en Cascadas de Haar que se explica detalladamente en el Capítulo 4 y en del que se muestra un ejemplo en la Figura 1.3 donde las Haar-Features han sido escogidos de manera que localizan elementos faciales con suma rapidez.
   Gracias a esto, es posible aplicar el algoritmo en tiempo real siéndo la propia aplicación la encargada de seleccionar la posición de los nodos de la malla facial y generando la deformación y cara media de manera autónoma.

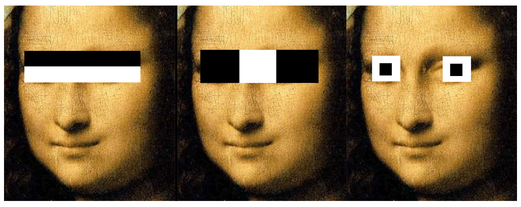


FIGURE 1.3: Monalisa. Ejemplo de la localización de *Haar-Features* sobre una cara desconocida.

5. **Algoritmo de reconocimiento.** Finalmente, para poder demostrar la validez de los algoritmos y resultados obtenidos, se implementó un método de reconocimiento facial basado en eigenfaces mediante PCA.

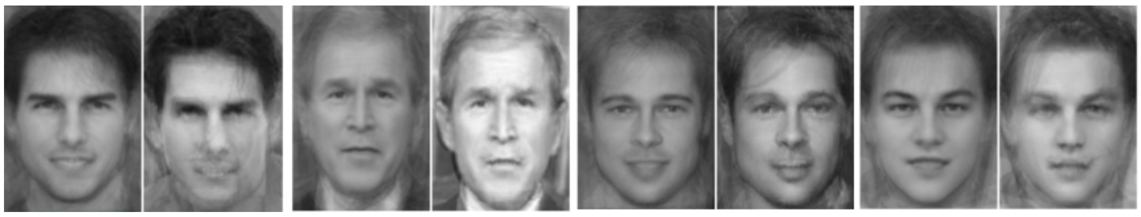


FIGURE 1.4: Ejemplos mostrando la cara media generada de manera manual (izda) frente a la generada de manera automática (dcha).

Una vez programada la aplicación, se pasó a estudiar y discutir, mediante un análisis estadístico basado en los resultados obtenidos, si la generación de las caras mediante deformación automática alcanzaba valores cercanos a la deformación manual comparando los índices de reconocimiento. El proyecto también incluye entre sus apéndices los resultados analíticos, imágenes y soporte, así como una completa guía del programa para ofrecer su posterior uso por cualquier persona interesada.Finalmente se incluye en código fuente completo de la aplicación para poder ser ampliado o modificado en futuras investigaciones similares.

El esquema general del algoritmo implementado se muestra la Figura 1.5

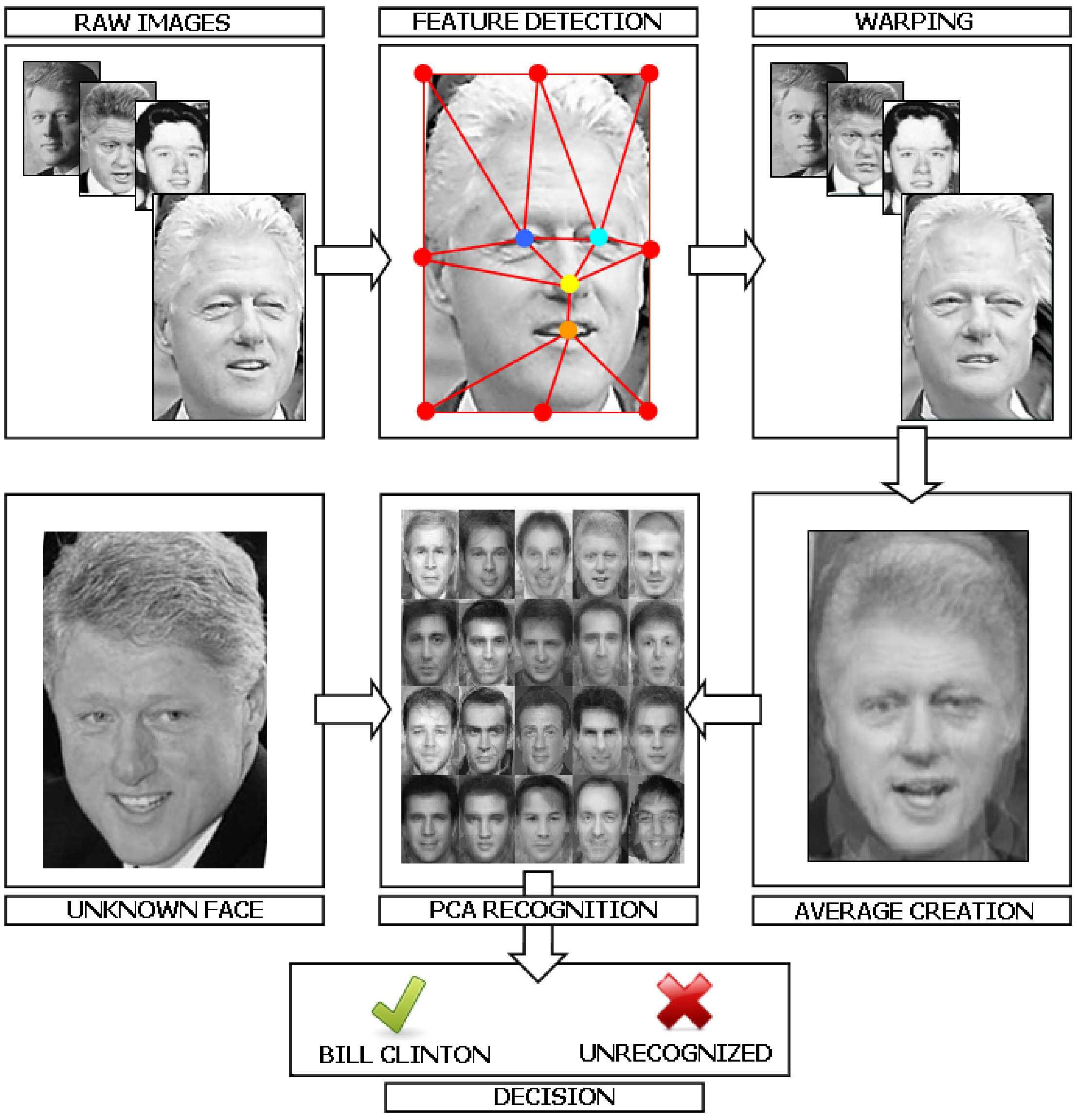


FIGURE 1.5: Esquema completo de la vida de la aplicación con todos sus pasos intermedios.

1.3 Resultados

Para completar el trabajo, se realizó un completo análisis de los resultados obtenidos, así como para reajustar parámetros y variables de los algoritmos con la idea de mejorar los resultados y adaptar el código a las necesidades encontradas durante el trabajo de investigación. Los datos se encuentran expuestos en el Capítulo 5. Para contrastar los resultados con programas externos se hizo uso de la página web `http://myheritage.com` que posee un motor de búsqueda facial comercial de código cerrado. Dado que comprar una licencia de un programa de reconocimiento facial se salía del presupuesto del proyecto, algunos de los experimentos fueron comparados con los expuestos por esta página bajo las mismas condiciones.

1.4 Publicación

Este proyecto fue seleccionado por la Universidad de Glasgow para optar a varias conferencias/galardones en Reino Unido:

1. **BMVC 2009** - British Machine Vision Conference 2009
2. **SET Awards 2009** - Premios al mejor tutor y estudiante por proyecto de investigación en la modalidad de Computer Science

Chapter 2

Introduction

2.1 Computer Vision

Computer face recognition is nowadays one of the most difficult tasks to be solved in terms of computer vision. During the last 10 years [1] this idea has been investigated and the results obtained have never been completely successful. In some cases, simple objects with straightforward shapes were identified correctly over carefully controlled backgrounds. In other cases like *Optical Character Recognition*, systems are able to detect and process texts with accuracies up to 95.38% [2].

Nevertheless artificial recognition has to deal with the real world and that is what makes this process extremely hard to be understood by machines. Faces are made up of wide varieties of lines, distances and shapes. As natural objects, human faces are formed by complex edges that make computers desperately difficult to approach them to any mathematical model. To make it more complex, human faces can change spontaneously through physical aspects like a beard or emotional expressions as a smile. Furthermore, the passage of time causes physical changes that barely can be predicted [3]. In spite of all these obstacles, face recognition appears to be a challenge not only for computers but also for human beings to teach computers how to deal with them.

2.2 Background

The earliest studies started in 50's by psychologists. Afterwards, some years later engineers also took part in the investigations. A few face expression studies were followed during the 60's. The search on automatic machine recognition faces happened over the past 30 years by psycho physicists, neuroscientists and engineers within research on various aspects of face recognition by humans and machines [4].

At the same time, engineers have tried to formulate the problem as recognition of 3D objects based on flat images in 2D. As a result, they adopted typical techniques of pattern classification where the face characteristics were used as attributes, obtaining semi-automated systems. (See the Bledsoe experiment for details [5]). Others focused the problem with subjective marks like ear's length or lip's thickness complicating much more the automation of the procedure.

In the 80's there was a surprisingly stop. Researchers were confused and all the results pointed a lack in technology support. Nonetheless during the early 90's, the commercial attention, technology advances and security demand encourage engineers and psychologists to continue the investigations extending the paths of interest in the matter.

During the last 15 years, the main target of the researches was to create a fully-automated system eliminating the difficulties due to face location inside an input image or the extraction of facial characteristics (eyes, mouth, nose, eyebrows, etc ...). In the meantime, important advances where reached in algorithms like *Eigenfaces* or *Fisherfaces* for face recognition and patterns detection.

2.3 Applications for the solution

Applications in face recognition are widespread. Focusing on security and control aspects, it would be very useful to automate the detection of people entering in a building, track the people that are entering in a plane or directly manage the people that take a walk in the street. This new concept of people control is being promoted by several countries as United Kingdom [6] or United States of America [7] due to terrorism attacks. On the other hand, face recognition can be very useful as a biometric method of identification to have access to any system like a computer or a building.

It is important to emphasize that face recognition is one of the leading fields in security and control environments. Several governments have invested huge quantities of money and time to investigate for applicable solutions. For instance, the US government has performed multiple evaluations to determine the capabilities and limitations of face recognition, and to encourage and direct future development. The *Face Recognition Technology Evaluation* (FERET) , sponsored from 1993-1997 by the *Defense Advanced Research Products Agency* (DARPA) was an effort to encourage the development of this technology [8]. Large firms like *Google Inc.* are also performing investigations in these aspects after the acquisition of *Never Vision Inc.* in 2006. For this reason, the face recognition improvement would be useful to improve actual applications in some fields:

- Security in airports, public places, main streets, underground ...
- Fast identification by the police or army of potentially dangerous people
- Preventative systems to control the access to computers or personal systems
- Web based solution as an Internet images search engine
- Web based solution as an entertainment service
- Artificial intelligence enhancement with the implementation in robots (human - robot interaction)
- Supplanting of the actual identification systems such as passports or ID Cards
- Decrease the rate of *Identity Theft*
- Video games, virtual reality, training programs
- Advance video surveillance, CCTV control, building controls, etcetera ...

Several enterprises like *L-1* offer devices installed at corporate buildings, banks, hospitals, airports and other kinds of entry points that scan faces and decide when you are welcomed [9]. Other companies like *My Heritage* (See Figure 2.2 and Figure 2.1) determine which celebrities you look like more or reveal if sons and daughters are more like their fathers rather than their mothers and vice-versa.

Another field of application is in automatic photo classification. That is the case of *iPhoto* by Apple [1], where the photo viewer software not only displays the personal album photos but also learns who are the people inside the pictures, permitting the user to view the photos that contains a specific person or groups of people using filters. This features, in conjunction with GPS tracking (Geolocation) permit to classify any personal image in terms of time and space, recording this data inside the image file header in a special format called EXIF [2]. Nevertheless, the implantation of these systems in our life, the idea of been controlled and located anywhere at any time should be treated consciously and with care.

[1]More details at `http://www.apple.com/ilife/iphoto/` last visited April. 10, 2009

[2]Standard definition at `http://www.kodak.com/global/plugins/acrobat/en/service/digCam/exifStandard2.pdf` last visited April. 10, 2009

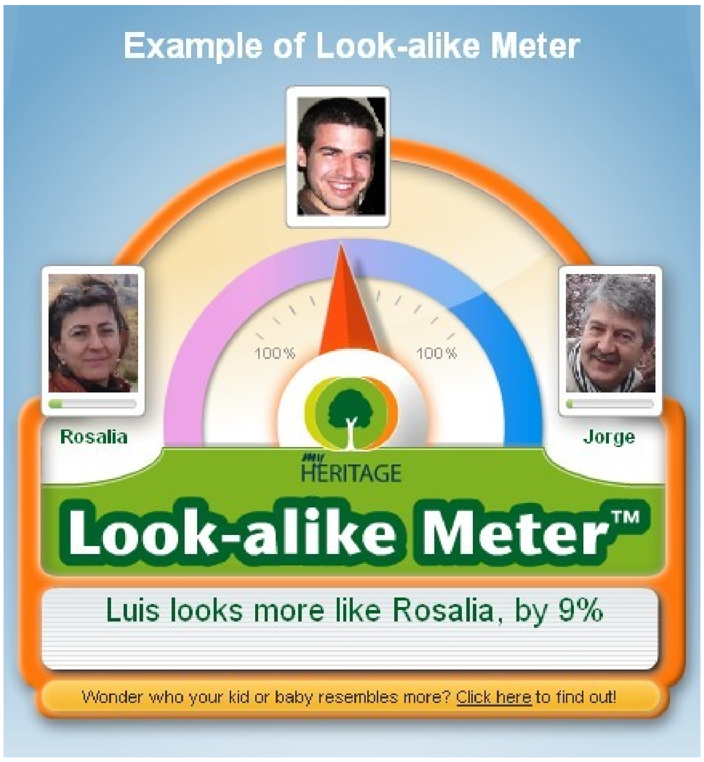

Example of Look-alike Meter
100%
100%
Rosalia
Jorge
my HERITAGE
Look-alike Meter™
Luis looks more like Rosalia, by 9%
Wonder who your kid or baby resembles more? Click here to find out!

FIGURE 2.1: Example of http://www.myheritage.com –*Look-alike meter*

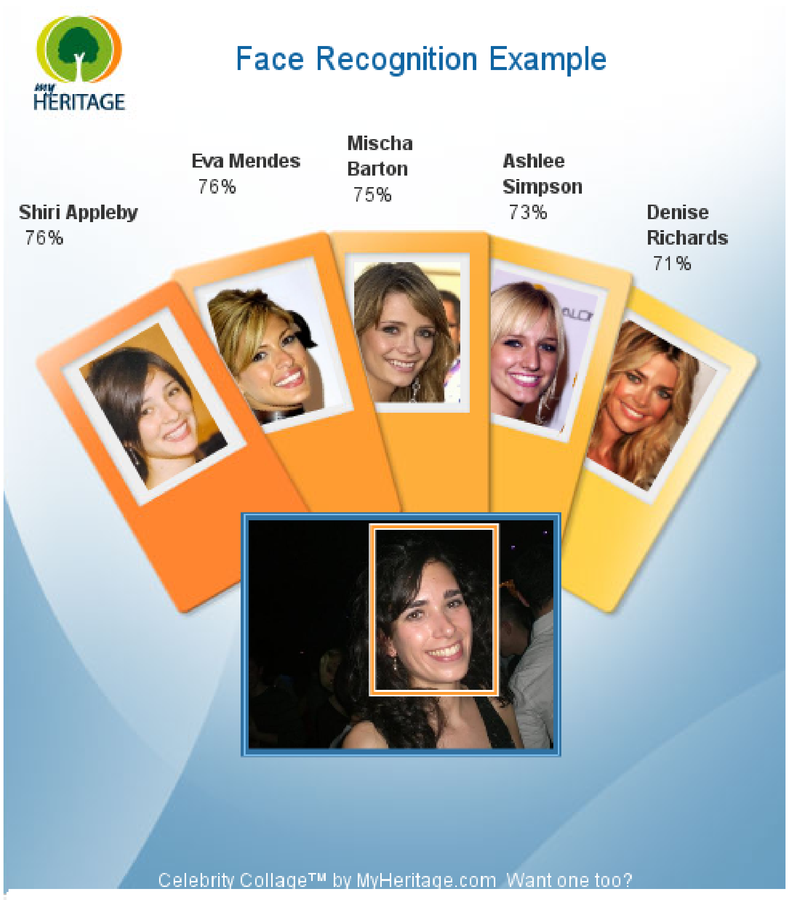


FIGURE 2.2: Example of http://www.myheritage.com –*Celebrity collage*

Chapter 3

Theoretical Aspects

3.1 Digital Image Processing

3.1.1 Introduction

An image could be defined as a bi-dimensional function *f(x,y)* representing a light intensity, where x and y are the spatial coordinates being the value of f in any point *(x,y)* denominated intensity or grey level. In case *f*, x and y are discrete values, the image will be called *Digital Image* [10]. Then, any digital image could be represented as a $M \times N$ matrix where each element is called *pixel*.

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & \cdots & f(0,N-1) \\ f(1,0) & f(1,1) & \cdots & f(1,N-1) \\ \vdots & \vdots & \ddots & \vdots \\ f(M-1,0) & f(M-1,1) & \cdots & f(M-1,N-1) \end{bmatrix}$$

Image analysis transforms any image in attributes (e.g. textures, distances, shapes or borders). To do that two tasks are required: **segmentation** (divide an image in small well delimited regions) and **description** (extract values that define the object uniquely). Finally, the target of the recognition technique is to interpret these objects, analyse their attributes and establish labels to those objects, making public this information to be used in following systems. To do that, the most extended method to determine which kind of object has been found is through a classifier, providing some intelligence to the machine.

3.1.2 Colour Spaces

Each one of the values of the matrix *f(x,y)* represent a colour, generally made of two components: Luminance[1] and Chrominance[2]. A colour space is a mathematical model that describes how to represent a colour just with numbers, typically as three components. During the following sections, the most used spaces will be introduced.

3.1.2.1 RGB Space

Most of the algorithms and applications are expected to deal with RGB colour space. That means, *Red*, *Green* and *Blue* components that define any colour can be separate independently in three identical dimension matrices. This model is based on a Cartesian Axis system, being the space a normalized cube where pure values red, green and blue are located in the vertices (1, 0, 0) , (0, 1, 0) , (0, 0, 1) respectively:

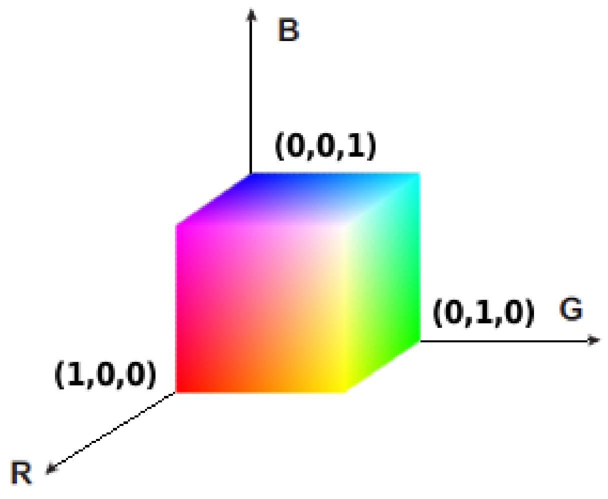


FIGURE 3.1: Three-dimensional representation of the RGB colour Space

In relation with the software developed for this project, it is important to notice that all the algorithms have been developed using the RGB space because of the simplicity it offers. In the other hand RGB space has a weak point: unexpected changes in ambient light will modify rapidly the intensity of the pixels. Thus, more robust systems could be used to avoid abrupt changes due to ambient light.

[1]Definition: Luminous intensity per unit area projected in a given direction
[2]Definition: Signal used in images systems to express the colour information of the picture, independently from the accompanying luminance signal

3.1.2.2 Grey Colour Space

One of the first applications of RGB space is the conversion into grey space. To convert a RGB pixel into a grey scale, there is not a correct conversion, depending always on the *human perception* and the *sensitivity response curve* of the camera used. If the three intensities have the same radiance in the visible spectrum (3.1)

$$f(x, y) = 0.333333 \times R + 0.333333 \times G + 0.333333 \times B \tag{3.1}$$

then the green will appear the brightest of the three due to the luminous efficiency function peaks inside the green region of the spectrum.

Several proposals have been discussed and approved, being two of them more used: Craig's approach (3.2) [11] and the model defined by the Commision Internationale de l' Eclairege (CIE) in 1931 based on human perception (3.3) [12] [13]:

$$f(x, y) = 0.300000 \times R + 0.590000 \times G + 0.110000 \times B \tag{3.2}$$

$$f(x, y) = 0.212671 \times R + 0.715160 \times G + 0.072169 \times B \tag{3.3}$$

The results applying the previous methods are displayed in Figure 3.2.

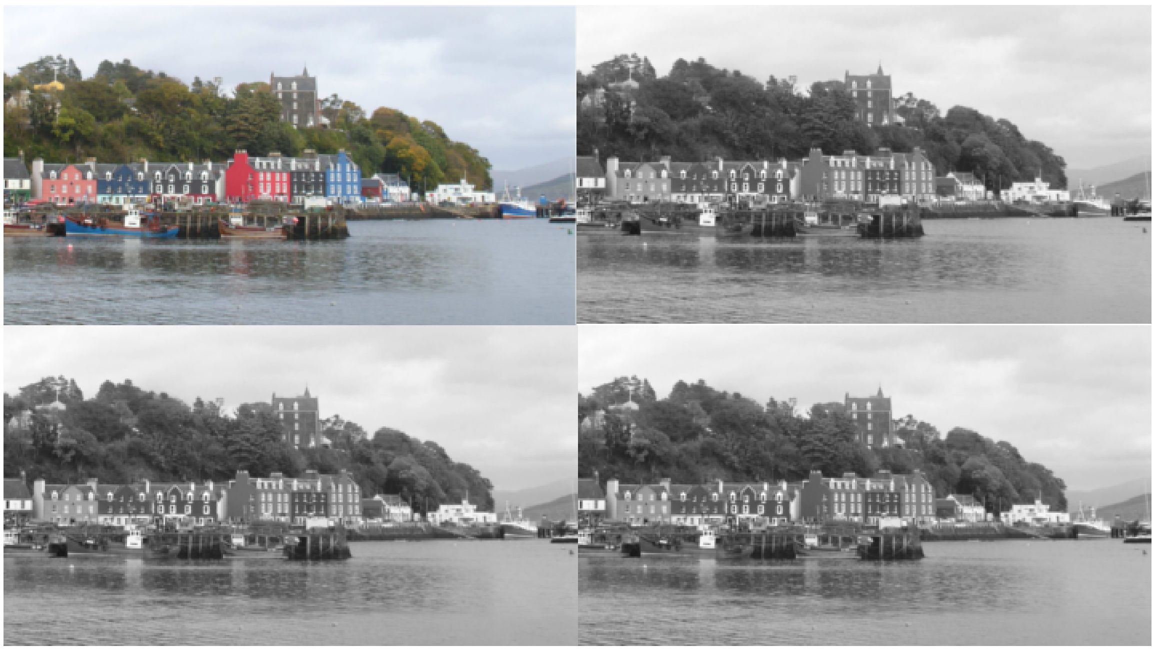


FIGURE 3.2: Examples of grey scales (Oban, Scotland 2008). Original picture, same radiance, Craig's and CIE models.

3.1.2.3 HSV Colour Space

The geometrical representation of this space is a cone (See figure 3.3). Vertical position corresponds to brightness (or value), radial position means saturation and angular position is for huge.[3] The main advantage of this system of representation is the abstraction of the hue component from the image colour, allowing intensity changes without modifying any colour saturation [13]. However, the problem is that the conversion from RGB to HSV and vice versa is not lineal (See Appendix B for further details). Images are by general rule stored in RGB format producing large computational cost if it is necessary to move into this space.

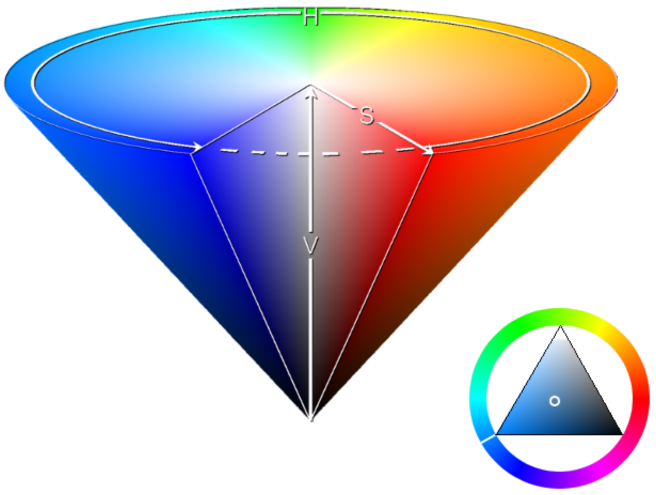


Figure 3.3: Selection of a colour using HSV model in Gimp

3.1.3 Mapping in digital images

There are several techniques for mapping an image. The classic two are *reverse mapping* and *forward mapping*. The difference between both is the way they are processed.

The first one is called forward mapping because the pixels are scanned and copied from the source image to the destination image taking into account the result of the warping function. The second one is called reverse mapping because for every pixel in the destination takes a value from the source image (see Figure 3.4). Thus, for this option to be valid the inverse of the warping function must exist (this is not always true, provoking holes or artefacts inside the image). The characteristic of reverse mapping is that it is ensured a value in all the pixels in the destination image while in forward mapping leaving some of the destination points unevaluated is possible.

[3]Hue can be normalized between 0 and 100% for some applications, but normally varies between $0^o \leftrightarrow 360^o$

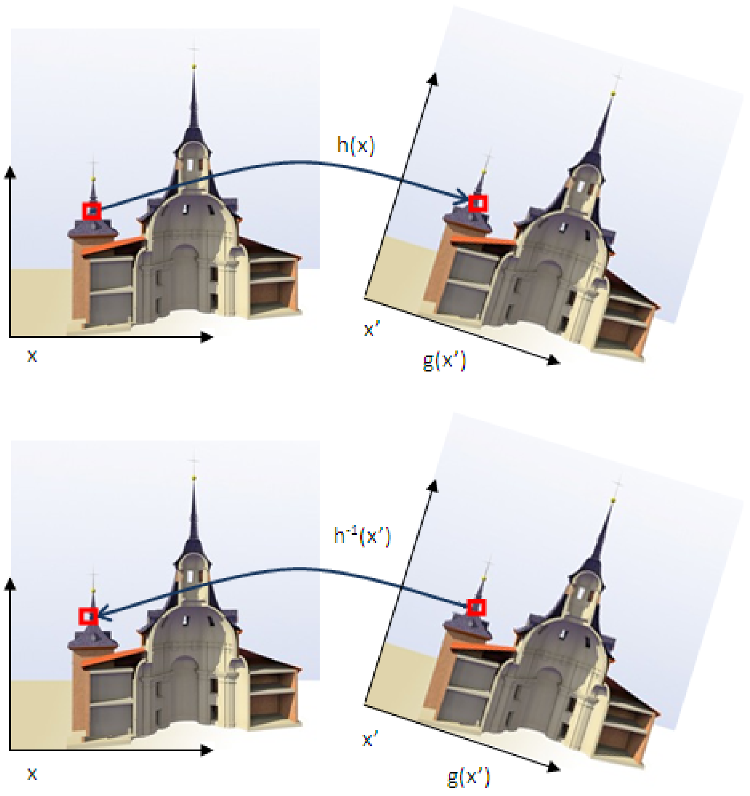


FIGURE 3.4: Mapping an image. Forward mapping in the upper image and inverse mapping in the bottom image.

1. **Forward mapping:**

   Send each pixel $f(x)$ to its corresponding location $x' = h(x)$ in $g(x')$

2. **Inverse mapping:**

   Get each pixel $g(x')$ from its corresponding location $x = h^{-1}(x')$ in $f(x)$.

Another procedure to map an image is to use Cross Segment Algorithm (CSA). This algorithm pays more attention to the relationship between the reference and the desired images [14]. Due to this method eliminates the searching cost, CSA became an accelerate method of inverse warping. However, sometimes is not possible to know the generalized disparity of the pixel in inverse warping and then the operation cannot be performed directly.

There have been proposed several optimized methods to avoid the problem of time and try to increase the speed [14]. All of them are classified in two ideas: increase speed according to the properties of the *epipolar line* or acquiring the generalized disparity of the desired image.

For this project application reverse mapping has been applied to make the process secure and establish a value in all the pixels of the image requiring as less resources in the machine as possible.

3.2 Image warping

3.2.1 Introduction

Image warping is the process which transforms an image into a new deformed image following a pattern or function. This process is actually used to recover distorted pictures projected into surfaces or to simply to create morphing effects for films or animations. Warping procedure estimate the new coordinates of an original pixel based on constraints defined beforehand.

Image warping has been used during last decades in medical imaging, computer vision, and computer graphics [15]. Main efforts have been focused on security, special effects for movies and scientific investigations. These deformations have become an interesting aspect because of the apparition of new computers capable of calculate tedious and difficult operations in real time. The fast increase in powerful machines, embedded devices and real time communications has made feasible the treatment of images, its processing and of course their recognition.

There are a wide variety of algorithms that provide warped images. All of them are based on geometrical relations between the pixel and the deformation function to be followed. Depending on the objective necessarily to reach, some are more recommended than others. The critical features of the image warping algorithms are speed, accuracy and CPU resources (algorithm's complexity). Because there are different geometric transformation techniques for digital images, a study of part of them has been done for the development of the application, and therefore it was decided which one fits better within the project requirements.

For this project, due to the quantity of images to be processed is quite high, in real time and the size of them might be adjusted, the best choice will be the faster assuming that the output image is truthful enough to be pre-processed later.

3.2.2 Basic transformations

The fundamental transformations involved in this deformations are three:

1. Scale by a factor **s**

   $x' = s \cdot x$

   $y' = s \cdot y$

2. Rotate by an angle θ

   $x' = x \cdot cos(\theta) - y \cdot sin(\theta)$

   $y' = x \cdot sin(\theta) + y \cdot cos(\theta)$

3. Shear by a factor h in one or both axis

$$X\text{-axis:}\begin{cases} x' = x + h \cdot y \\ y' = h \cdot y \end{cases} \qquad Y\text{-axis:}\begin{cases} x' = h \cdot x \\ y' = y + h \cdot x \end{cases}$$

Each one of them is based on a different function that describes the destination axis (x', y') for every location in the source (x, y). This method can be applied vice versa if the lineal application is considered invertible.

These are the basic functions, but there are infinite equations that can be applied to create almost any kind of deformation in the destination picture taking values of the original pixels. In Figure 3.5 some samples are displayed.

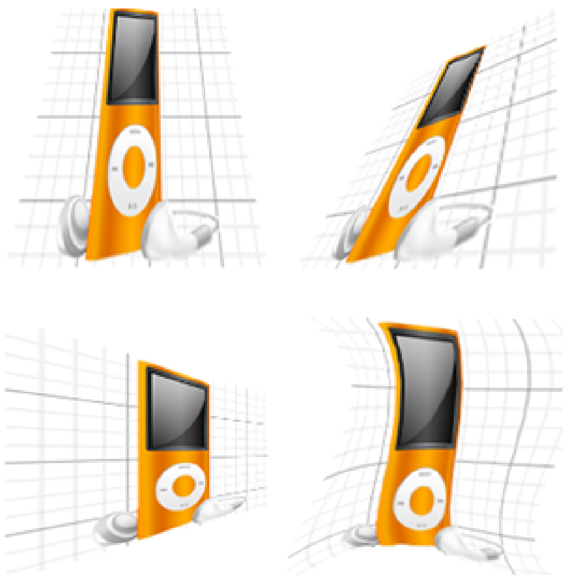


FIGURE 3.5: Warping examples over the same image: Shear, affine, perspective and waves.

3.2.3 Actual algorithms for image warping

3.2.3.1 Transformation with one pair of lines

If exist a pair of lines, the first one referred to the source image and the second one referred to the destination image, any pixel in the source image can be transformed based on a mapping relation between both lines (See Figure 3.6). This algorithm has been extracted from [15], [16].

If X' and X are any pair of coordinates in the source picture and destination picture respectively and $P'Q'$ and PQ are the lines defined by extreme points in the source and destination pictures respectively, it is possible to define the following equations:

$$u = \frac{(X - P) - (Q - P)}{\|Q - P\|^2} \tag{3.4}$$

$$v = \frac{(X - P) \cdot Perpendicular(Q - P)}{\|Q - P\|^2} \tag{3.5}$$

$$X' = P' + u \cdot (Q' - P) + \frac{v \cdot Perpendicular(Q' - P')}{\|Q' - P'\|^2} \tag{3.6}$$

where in 3.5 and 3.6 *Perpendicular()* returns the vector perpendicular to, and the same length as, the input vector in the argument. The value u corresponds to the position along the line PQ while v is the distance from the line to the pixel in the source image. The range of values of u goes between 0 and 1 as long as the pixel moves from P to Q and it could exceed these values outside this range. The value v corresponds to the perpendicular distance from the position of the pixel to the line PQ.

This option applies to each pixel of the whole image a coordinate transformation by rotation, translation and/or scale. The scale will be done only in the direction of the line while the rotation and the translation will depend on the coordinates of the pixel.

The method to be used with this algorithm 3.1 is described as the following:

Algorithm 3.1: One pair of lines warping procedure

```
1 foreach $pixel\ X\ in\ Image_{destination}$ do
2     find $u, v$ ;
3     find $X'$ in Image$_{source}$ for the obtained $u, v$;
4     copy $pixel(X)_{destination} = pixel(X')_{source}$;
```

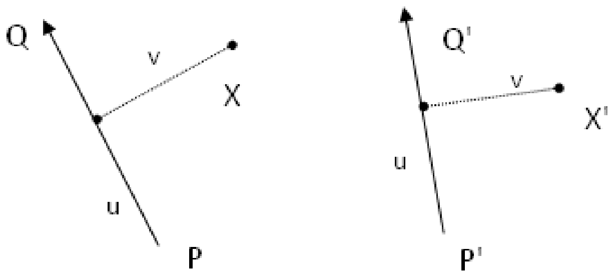


FIGURE 3.6: Single pair of lines method. Description of the variables involved

Due to this process is done using reverse mapping, it ensures that all the pixels in the destination image will be defined one by one by any of the pixels in the source image. Some examples of this algorithm have been displayed in the Figure 3.7.

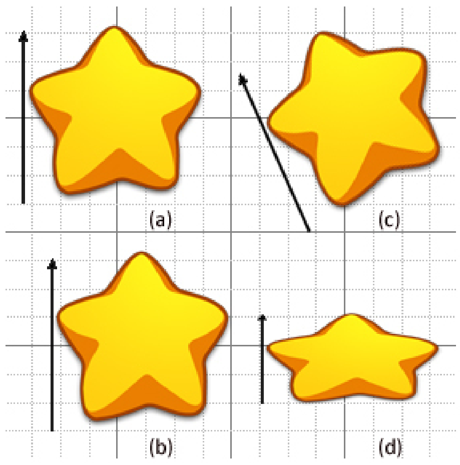


FIGURE 3.7: Single pair of lines method. Operations applied to each pixel. Original, displacement, rotation, scale.

Since faces contains more than a single line to control all the relevant facial features (outlines, chin borders, eyes's lines, etcetera ...), it is necessary to implement this method for multiple lines. This new improvement is described in the following section.

3.2.3.2 Transformation with multiple pairs of lines

The big innovation for this method consists on the management of multiple lines during the morphing process. In this case, the interaction of each pixel with all the lines will be appreciated. As [16], [17] explains, the closer the distance between the pixel and any line, the bigger would be the interaction between them. To fix this problem a weighting of the coordinate position for each line is applied following the Equation 3.7.

$$weight = (\frac{length^p}{a + distance})^b \tag{3.7}$$

Where *length* is the longitude of each line, *distance* is the distance between each pixel and that line and *a, b* and *p* are constants to determine the finest approach. The values of these constants have been studied and *bounded* in Table 3.2.3.2

Parameter	Limits	Explanation
a	0-0.1	If a is near zero weight $\rightarrow\infty$ and points on the line will be displaced exactly in the direction of the line
b	0.5-2	Establish the relative *strength* between the lines decline with the distance
p	0-1	Relative strength of the lines depending on its length. The longer the line, the stronger it is if p equals one

TABLE 3.1: Description of parameters in multiple lines

The multiple lines algorithm is described fully in Algorithm 3.2

Algorithm 3.2: Multiple lines warping procedure

```
foreach pixel $X$ in Image $_{destination}$ do
    set DSUM = (0,0);
    set weightSum = 0;
    foreach line $\overline{P_iQ_i}$ do
        calculate $u$ and $v$ based on $\overline{P_iQ_i}$ ;
        calculate displacement $D_i = X_i' - X_i$ for line $\overline{P_iQ_i}$;
        calculate distance = shortest distance from $X$ to $\overline{P_iQ_i}$;
        calculate weight = $(\frac{length^p}{(a+distance)})^b$;
        set DSUM = DSUM + $D_i \cdot$ weight;
        set weightSum = weightSum + weight;
    set $X' = X + \frac{DSUM}{weightSum}$;
    copy $pixel(X)_{destination} = pixel(X')_{source}$;
```

The graphical representation of the method helps for the understanding of the algorithm. In the Figure 3.8 the illustration on the left represents the coordinates in the destination while the right one represents the source.

X' is the point we are sampling from the source picture and X is the destination position. This new point is computed with the weighted average of the two pixel locations X_1' and X_2'. The resulting graphic of this method reveals that points in the same line or very close to them are moved in the same direction as the line and pixels further from one a specific line are affected by each one of them. Some examples of this algorithm are illustrated in the next Figure 3.9.

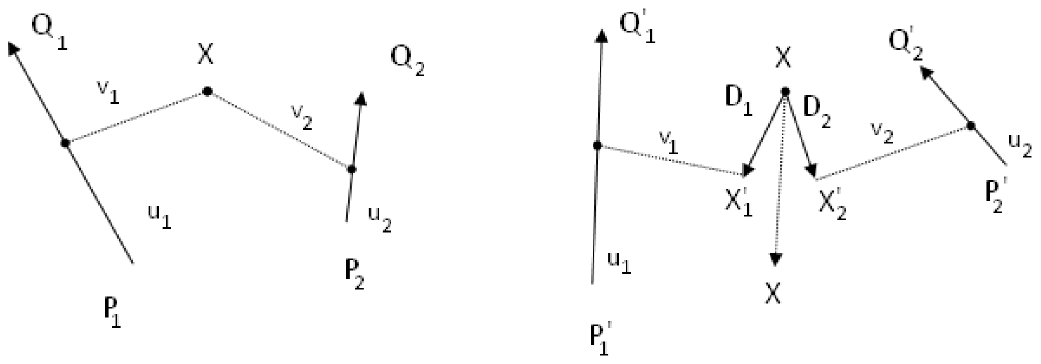


FIGURE 3.8: Multiple pairs of lines method. Description of the variables involved.

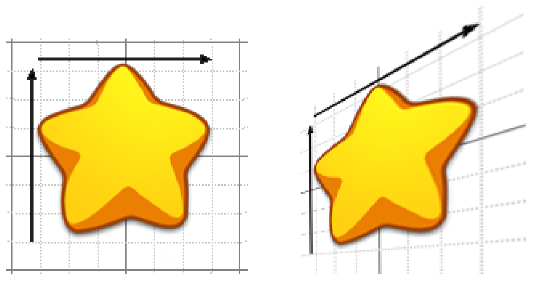


FIGURE 3.9: Multiple pairs of lines method. Description of the variables involved.

3.2.3.3 Triangular warping. Barycentric coordinates

The principle of this method is to create deformed pictures based on a triangular mesh. It is based on *piecewise polynomial transformations.* Supposing that the original and final images have attached a mesh that define the location of the most significant features and both meshes are formed by simple triangles, the final mesh's triangles can be supposed as a change in the coordinates of the corresponding original triangles.

The following method can be applied hold up by the thought that calculating the barycentric coordinates of the final mesh for each pixel, makes possible to obtain the equivalent location of the pixel in the original triangle. Then, the alteration of the position and colour of all the pixels included inside one triangle would be warped to the final shape. So, with this method is possible to find for each triangle in the warped mesh an equivalent in the original one and copy its intensity value to the warped image through a *triangle to triangle correspondence* [18].

Considering a triangle T defined by its three vertices A, B and C, any point P situated inside the triangle is considered as:

$$P = \alpha \cdot A + \beta \cdot B + \gamma \cdot C \tag{3.8}$$

Where α, β and γ are the barycentric coordinates, that also fulfil the equation

$$\alpha + \beta + \gamma = 1$$

To obtain the barycentric coordinates of any point $P(x, y)$ in a given triangle with vertices $A(x, y), B(x, y)$ and $C(x, y)$):

$$P_x = \alpha \cdot A_x + \beta \cdot B_x + \gamma \cdot C_x$$

$$P_y = \alpha \cdot A_y + \beta \cdot B_y + \gamma \cdot C_y$$

where $\gamma = 1 - \alpha - \beta$

$$P_x = \alpha \cdot A_x + \beta \cdot B_x + (1 - \alpha - \beta) \cdot C_x$$

$$P_y = \alpha \cdot A_y + \beta \cdot B_y + (1 - \alpha - \beta) \cdot C_y$$

Rearranging the terms:

$$\begin{pmatrix} \mathrm{P}_x - C_x \\ P_y - C_y \end{pmatrix} = \begin{pmatrix} \mathrm{A}_x - C_x & B_x - C_y \\ A_y - C_y & B_y - C_y \end{pmatrix} \cdot \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \tag{3.9}$$

That is

$$M = \begin{pmatrix} \mathrm{A}_x - C_x & B_x - C_y \\ A_y - C_y & B_y - C_y \end{pmatrix} \Longrightarrow \begin{pmatrix} \mathrm{P}_x - C_x \\ P_y - C_y \end{pmatrix} = M \cdot \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \tag{3.10}$$

Where the barycentric coordinates are obtained as:

$$\begin{pmatrix} \alpha \\ \beta \end{pmatrix} = M^{-1} \cdot \begin{pmatrix} \mathrm{P}_x - C_x \\ P_y - C_y \end{pmatrix}$$

$$\gamma = 1 - \alpha - \beta$$

With this information is possible to deform the pixels included in each one of the triangles in the original mesh to the required shape. The computational cost is low because the matrix to be inverted is just a 2×2 matrix that would never be *singular* if the triangle is well defined. This method could be defined as an step between multiple lines deformation and the next method, mesh warping.

3.2.3.4 Mesh warping

This process was pioneered by Douglas Smith for using it in the film *Willow, 1980* [15]. The process followed in this method is based in changes on points of different meshes instead of lines (see [15], [19] for more details). The initiative is to break up both images into small sections that are plotted onto each other deformed for the morph process. In order to apply this method, two arrays are needed. The first one contains the source coordinates of the mesh while the other one encloses the destination mesh. It is compulsory that both matrices have the same size to facilitate the one-to-one relation for the needed computations.

Two passes are needed to create the final image warped. This method is called two-pass warp. During the first pass the image is only warped vertically, but in the second pass the image is full transformed with a horizontally warp over the pre-warped image done before. Occasionally, in some parts of the picture could be recommended to do the first the horizontally to maintain the picture quality [20].

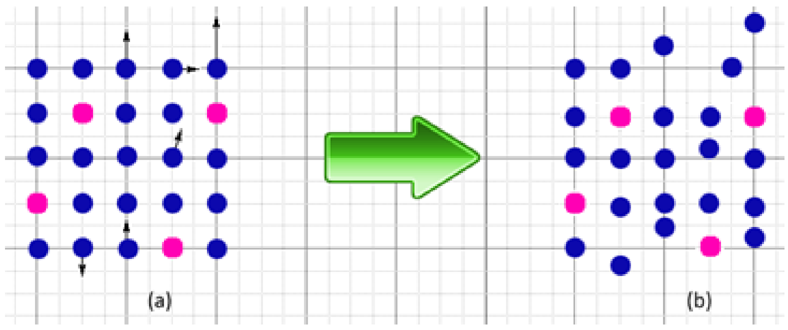


FIGURE 3.10: Mesh warping. Operations applied to each pixel.Original mesh (a) and desired mesh (b)

If the original image is called I_S and the destination image is called I_D. There are two meshes defined as M_S and M_D that contains the coordinates of both meshes (See Figure 3.10). It is important to notice that both meshes are constrained to be topologically equivalent [15]. That means that no folding, discontinuities or self-intersection between nodes in both networks are allowed. During this process intersection some intermediate images are created. The Algorithm 3.3 represents the steps to be followed to implement this method.

Algorithm 3.3: Mesh warping procedure

```
1 foreach frame f do
2     interpolate linearly mesh $M$ between $M_S$ and $M_D$ warp image $I_S$ to $I_1$ using
      meshes $M_S$ and $M$;
3     warp image $I_T$ to $I_2$ using meshes $M_T$ and $M$;
4     interpolate linearly image $I_f$ between $I_1$ and $I_2$
```

Interpolation of the meshes

There are several methods to implement the interpolation between two collections of coordinates. But, the most used method is the *bi-cubic spline* interpolation. More precisely, the usual algorithm is Catmull – Rom spline interpolation. A deeper explanation is attached at the end of the text for further analysis. (See Appendix A)

An example of the deformation created by using this method is displayed in Figure 3.11 based on the example of [21]. In this case the deformation has been created following a cylindrical pattern as a destination mesh.

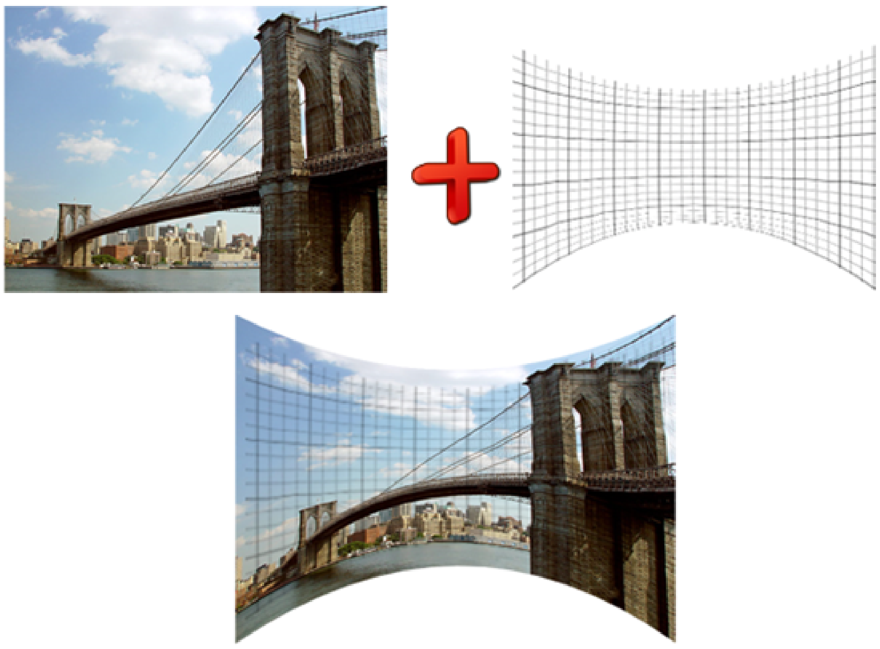


FIGURE 3.11: Mesh Warping. Example using a cylindrical destination mesh. Brooklyn Bridge (New York City, US), desired mesh with a cylindrical deformation applied and resulting picture once the warping is executed

To obtain a smooth and continuous batch of pictures in the transition between both pictures is recommended to apply a cross-dissolve filter[4]. This action will provide a natural behaviour in the path between the original image and the final result.

3.2.3.5 Selection of a warping algorithm for the project

After the previous review of the different deformation algorithms, two of them were implemented for this project: Transformation with multiple pairs of lines and triangular warping. The results given by both methods are described in Chapter 4, being the

[4]That means, a simple linear transition between images making a lineal change

second method more stable and robust than the first one for the aims followed in the study. The reason is mainly because of the theoretical concept of each method: Multiple pairs of lines is more focused obviously on lines that define boundaries in the face while triangular mesh warping is based on points that define a deformed space.

Because the target here is to make the process to work automatically and the actual methods of pattern recognition estimate positions rather than contours, it is easier to create a mesh with all these automatically detected positions instead of define those regions by lines (normally the outlines of the face are not straight lines but curve lines). On the other hand, mesh warping could be the "next step" after triangular mesh method. It requires large number of nodes (much more than triangular mesh method) in order to create a smooth surface defining small variations between points. This method offers a high continuity between the small sections defined by the nodes but due to the level of complexity that it demands and the computer load work that it consumes, triangular mesh warping comes up as the best choice.

Therefore, triangular mesh warping algorithm is implemented and explained in detail later. All the results obtained in Chapter 5 are based on deformations using that method.

3.3 Objects detection

3.3.1 The problem. How to recognise objects?

Image analysis for object detection is concerned with the extraction of measurements, data or information from an image by automatic or semi-automatic methods. By definition, the systems to be analysed are not limited to the classification of scene regions to a fixed number of categories, but they are rather designed to provide a description of complex scenes whose variety may be enormously large and ill-defined in terms of a priori expectation [22].

The objective in this study will be to find as fast and accurate as possible facial patterns defined beforehand: both eyes, nose and finally the mouth for each face. It is important to take into account that the application developed starts from the idea that any input image contains faces but it could be changed simply to any other static object, converting it into a portable application for future projects.

The final usage for objects detection would be face deformation. Therefore, depending on the number of objects detected inside the face, the deformation will vary, being better as long as this quantity is increased. One of the requirements for this project is a *pseudo* real-time analysis. That means that the results must be shown *just-in-time* fast and ready to be used afterwards by other systems. That is the reason why the method selected was to use *Rapid Object Detection* through a *Boosted Cascade* of Simple Features, described in the next section.

3.3.2 The solution. Haar Cascades

Due to human faces are complex objects it is complicated to find features or heuristics that could confront the huge variety of instances of the object class (e.g faces, eyes, mouths ...) that may rotate in any direction, captured in different light conditions or the simple apparition of glasses, beards or wrinkles. For such a objects, statistical models (here called classifiers) may be trained and used to detect the desired targets.

To do that, statistical models will be taught using multiple instances of the object to be recognized (these instances are called *positive*) and also multiple samples of *negative* instances where the object does not appear. The collection of all these samples, positive and negative, form a *training set*. During the training process, face features will be extracted from the training set and unique features of each image will be used to classify the object. It is important to remark that using this method, if the cascade does not detect an existing object it is possible to add the sample to the classifier training set and correct it for the next time.

The statistical approach used in this project has been defined using the OpenCV libraries (mentioned and explained later on during the following sections) based directly on the Viola & Jones publication [23]. This option applies simple *Haar-Like* features and a cascade of boosted tree classifiers as a statistical model. The classifier must be trained on images of the same size and detection is done using a window of that size moved along the whole picture. For each step, the algorithm check if the region *looks like the desired object* or not. Furthermore, to include possible sizes of the images to be detected, the classifier has the ability to scale the patterns. To make this method works, it is necessary just a batch of Haar-like features and a large set of very simple classifiers to classify the image region as the *desired object* or as a *non desired object*. Each feature is determined by the shape of the feature, its position relative to the search window origin and the scale factor applied on the feature. In total, 14 templates shown in Figure 3.12 are used for the project module.

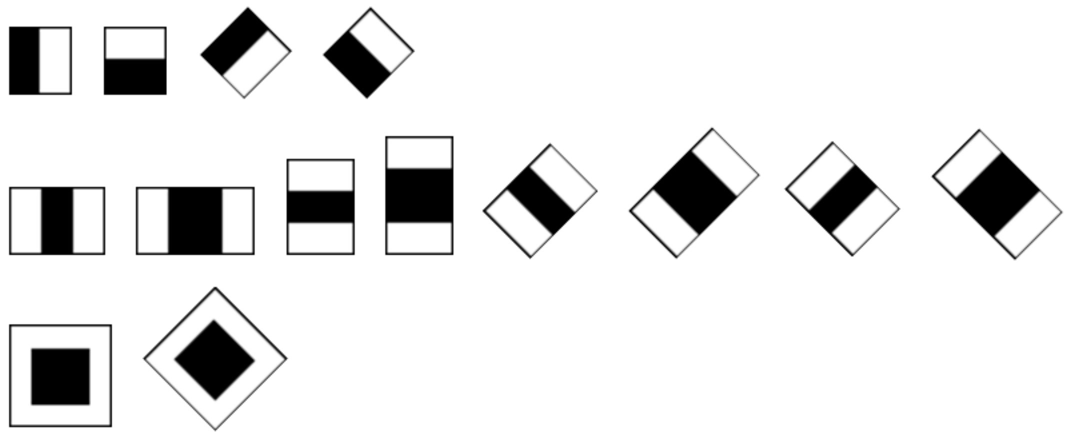


Figure 3.12: Set of Haar-like templates used for object detection. Edge features, line features and centre-surround features.

As in the previous Figure, each feature is designed using two or three *black* or *white* rectangles horizontal, vertical or rotated by 45^o. To compute the Haar feature's value, just a weighted summation of two components is needed: the pixel sum over the whole area of the feature and the sum over the black rectangle. Once this simple calculation is done, the weights of both components are of opposite signs and they are normalized dividing the summation over the black rectangle by the total area. As an example, for the second feature:

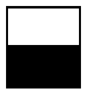


Figure 3.13: Example of Haar-like templates used for object detection.

$$2 \times weight_{black} = -4 \times weight_{whole} \tag{3.11}$$

or for the thirteenth feature:

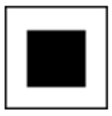


Figure 3.14: Example of Haar-like templates used for object detection.

$$weight_{black} = -9 \times weight_{whole} \tag{3.12}$$

Now, instead of computing directly the pixel sums over multiple rectangles and make the process of detection incredibly slow, [23] defined a way to make the summations faster: *Integral Image.*

$$ii(x, y) = \sum_{x'<x, y'<y} i(x', y') \tag{3.13}$$

where $ii(x, y)$ is the integral image and $i(x, y)$ is the original image. The summation of pixels over a single window $r = \{(x, y), x_0 \leqslant x \leqslant x_0 + w, y_0 \leqslant y \leqslant y_0 + h\}$ can be done using the surrounding areas as showed in Figure 3.15.

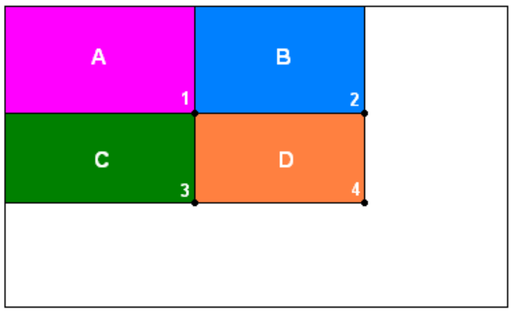


FIGURE 3.15: Area of different regions of pixels.

where the sum of the pixels of the rectangle D can be computed with four array references as is demonstrated in [23]. The integral image at 1 corresponds to the summation of all the pixels included in rectangle A. Just as the previous example, the value at 2 is $A + B$, the value at 3 is $A + C$ and finally the value at 4 is $A + B + C + D$. Therefore, the sum within D can be done as $4 + 1 - (2 + 3)$. That means that the pixel sum over a rectangle can be done regardless of the size, just taking into account the corners of the rectangle.

$$RecSum(r) = \underbrace{ii(x_0 + w, y_0 + h)}_{4} + \underbrace{ii(x_0, y_0)}_{1} - \underbrace{ii(x_0 + w, y_0)}_{2} - \underbrace{ii(x_0, y_0 + h)}_{3}$$

If a decision tree classifier is created taking into account each one of the feature value computed over each area of the image, as in 3.14 for two terminal nodes or 3.15 for three, where each f_i will give $+1$ if the obtained value is inside a threshold pre-defined and -1 otherwise.

$$f_i = \begin{cases} +1, & x_i \geqslant t_i \\ -1, & x_i \leqslant t_i \end{cases} \tag{3.14}$$

$$f_i = \begin{cases} +1, & t_{i,0} \leqslant x_i < t_{i,1} \\ -1, & else \end{cases} \quad (3.15)$$

where $x_i = w_{i,black} \cdot RecSum(r_{i,black}) + w_{i,whole} \cdot RecSum(r_{i,whole})$

It is important to notice that each classifier is not ready to detect an object by itself. It notices simple feature in the image (like a border or a point). For instance, the eye region is often darker than the cheeks, or the iris is darker than the rest of the eye. (supposing a correct size and orientation of the feature as in Figure 3.16)

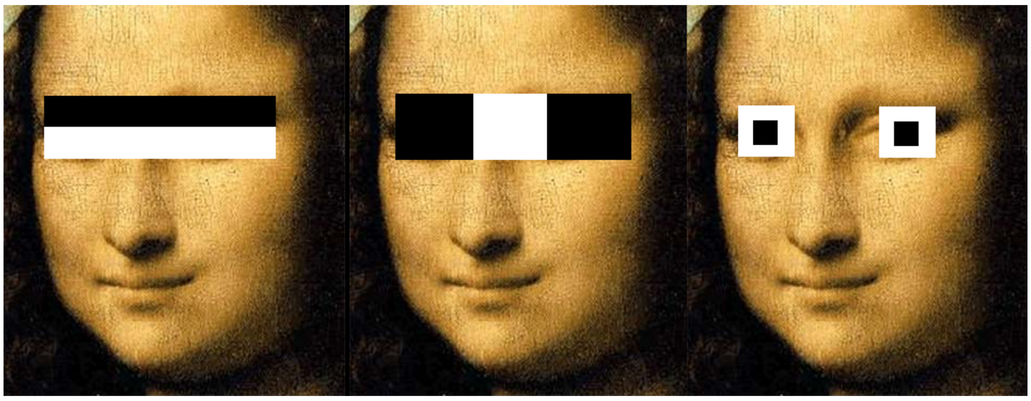


FIGURE 3.16: Monalisa. Examples of simple features over a known face.

After the creation of these classifiers (called *weak classifiers*) a list of complex robust classifiers is built out with the union of all the weak classifiers iteratively as a weighted sum of weak classifiers, being each one increasingly more complex. Afterwards, a cascade is created where first positions are for simple classifiers and final positions for the most complex. As far as the window is scanned by each classifier, it can be rejected or sent to the next F_i as in Figure 3.17 is explained.

3.3.2.1 Selection of a object location algorithm for the project

Because Haar Cascades are based on statistical analysis, this method provides fast and reliable results and it is not necessary to follow a learning process for the algorithm because there are public trained databases for any feature (explained after in Chapter 5). These advantages make Haar Cascades the best option to implement the complete location program.

The face database used for this project contains images of people in different conditions and situations but all of them have in common that are portraits. That means that face

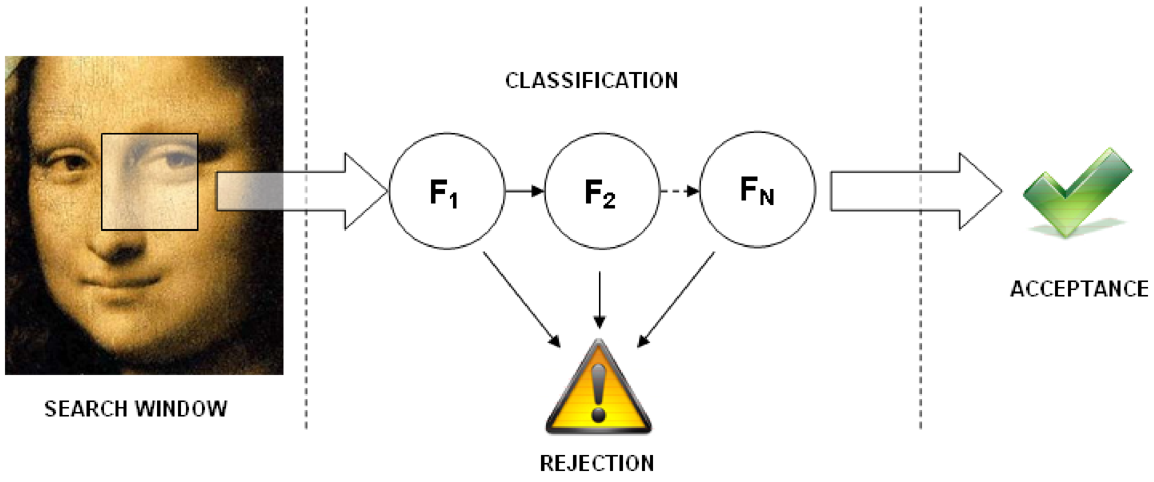


FIGURE 3.17: Cascade of Haar. Each classifier can reject or pass to the next classifier the image.

location is not a big deal because input images are just faces, removing the necessity of looking for faces inside full body pictures. Despite of that, the application is ready to recognise faces inside images and afterwards recognise internal features (eyes, mouth and nose) making the solution wider and more complete.

3.4 Face recognition

3.4.1 The location problem

Most of the techniques used for facial recognition assume the availability of *ideal images* (good light location and intensity, uniform background, smooth variations in the object's shape, sufficient resolution, uncompressed, etcetera . . .). However, real scenarios are completely different: poor defined shapes, diffuse or moved objects and complex backgrounds that force the use of previous processing (pre-processing) to acquire a better image ready to extract face regions from the environment.

Probably this is one of the most exciting and difficult tasks in face recognition: to extract face coordinates with low probability of failure. Until the middle 90's the face was searched under highly controlled backgrounds (one shade). These methods included the use of neuronal networks, skin texture or deform meshes based on facial characteristics.

3.4.1.1 Face location through *global* representation

Among all the actual techniques used to detect faces the one that has been mostly emphasized is the rules codification that describes the characteristics of the face and its relations (relative distances, positions or even areas described by a group of points). This method presents two particularly difficulties: It is necessary to know beforehand that there is a face in the scene and secondly the generation of too much false positives in complex backgrounds [24].

Other alternatives have been studied to improve accuracy and quality rates like global techniques based on localization of faces treating them as a pattern location problem assuming that the face is the pattern. When the templates are used, a pattern is searched defined manually in the input image. That is, *face* and *non face* categories are determined by an expert. Methods based on appearance learn the templates from the training images. In general, these methods use statistical analysis and automatic learning to find the relevant features of the categories *face* and *non face*.

The disadvantage of algorithms based on templates is the difficulty for generalize the algorithm to be used in different situations of images (illumination, spatial hiding or position). To fix that, the expert have to define a high number of templates to cover any possible situation and therefore represent any case. Both methods develop an exhaustive search of patterns for any position or scale extracting the output region and classifying the result with the classical methods.

The last group of methods provides the face location based on colour distribution techniques. Because faces maintain a narrow colour band inside the colour space and most information is concentrated in chrominance and not in luminance, this component can be rejected using segmentation models: *RGB, HSV, YCrCb, YIQ, CIE, XYZ* [25]

3.4.1.2 Face location through *local* representation

These methods try to locate faces detecting facial characteristics like eyes, eyebrows, mouth, face contour, etcetera... and then they are combined with other proceedings that identify the relations and verify the existence of the face. These procedures are based on the idea that humans are able to recognize faces in different positions or luminance situations because there should be a group of distinctions or properties independent to those variables.

3.4.2 Face recognition algorithms

3.4.2.1 Eigenfaces. Principal Components Analysis

The eigenfaces method is probably the simplest and efficient method used for human face recognition. Contrary to others algorithms based on the identification of gestures and classification of distances between them, this method evaluates the image as a whole. Thanks to this consideration, is feasible to use not only in prepared environments with pre-defined light conditions but also outside, in the real world. Basically, this method is supported on a simple concept: reduce useless information as much as possible. When a picture is studied, even if it is small, there is a lot of information expressed inside. Therefore, depending on the final purpose, the focus on some parameters will be highlighted over the rest.

When a general image is being analysed, is expected that most of the area of the image will not represent a face. That is why is necessary a method to detect and extract human faces from the rest of the background as a fast and accurate procedure. The way to do that is creating a base of faces and trying to represent any analysed image as a combination of them. It is easy to compare this technique to the colours representation. The base of colours is formed by primary red, green and blue. Thus, it is possible to represent any colour as a partial addition of red, green and blue. For instance, orange will be formed as a combination of the maximum value of red, half value of green and nothing of blue.

Referring to eigenfaces, the problem becomes to find the precise collection of base faces that represents the best a specific face. Taking advantage of the previous comparison, the issue is how much quantity of red, blue and green paint does the painter need to reproduce the colour he is already watching. It is important to notice that the precise selection of the base faces will provide better results in the recognition step. The creation and development of the mentioned *face base* is explained in detail in several publications [26], [27], [28], taking into account that some information reduction has to be applied in order to diminish the complexity of the original problem up to 1/1000.

Each face we want to introduce in the base to be classified can be projected into a *face space* and then analysed as a vector. After that, any existent distance method like Euclidean distance, k-nearest-neighbour, Mahalanobis distance or neural network can be used to classify the input face. The problem is solved after this step, inducing next points to be effortless.

The eigenfaces method can be divided into three differentiated sections:

- Generation of eigenfaces.
- Projection of trained data into face-space to be used afterwards with a classification method.
- Evaluation of a projected test image and compare it with training data.

3.4.2.2 Linear Discriminant Analysis

Linear Discriminant Analysis is a statistical approach for classifying samples of unknown classes based on training samples with known classes. The purpose of this technique is to maximize between-class (e.g., across users) variance and minimize within-class (e.g., within user) variance. If this is moved to high dimensional face data, this method faces the small sample size problem that arises where there are a small number of available training samples compared to the dimensionality of the sample space [29].

3.4.2.3 Elastic Bunch Graph Matching

Elastic Bunch Graph Matching is based on the idea that human face images have many non-linear characteristics that are not supported by the linear analysis methods such as variations in pose (standing straight versus leaning over), face expression (smile versus sad) or illumination (outdoor lighting versus indoor fluorescents).

A Gabor wavelet transform (see [30] for more details) generates a dynamic link architecture that projects the face onto an elastic mesh. The Gabor jet is a node on the elastic grid (represented by circles) which determines the image behaviour around a given pixel. It represents the result of a convolution of the face image with a Gabor filter, which is applied to detect shapes and to extract relevant features by means of image processing. The recognition process relies on the similarity of the Gabor filter response at each Gabor node. This biologically-based methodology that processes Gabor filters is a procedure executed in the visual cortex of higher mammals. The main disadvantage of this method lies on the necessity of use an accurate landmark localization, that often can be achieved mixing PCA and LDA methods.

3.4.2.4 Selection of a face recognition algorithm for the project

For the programming of the application in this project, face location by global representation was chosen as the best option. This decision was taken first of all because input images contain faces occupying almost all the image region and secondly because PCA method was already implemented and tested making the programming less difficult and more stable and accurate. Furthermore, eigenfaces is the most simple implementation and the one used as the basis in complex software applications by specialized companies and is the one that investigators use for their publications.

Chapter 4

Development of the application

4.1 Language, libraries and environment

4.1.1 Programming Language: C++

For the complete development of this project, C++ language has been chosen. One of the main requirements was to increase speed as much as possible, and C++ is ideal for high performance applications because it works not only with high level functions but also with low-level machine functions. C++ is a *middle level language* object oriented (OO) with some features like classes management, class heritage and overloaded functions. In the Figure 4.1 a comparison between C and C++ is performed showing that for small-medium applications C++ is recommended, being better C for big applications or high resources necessities [31]. For this project `gcc v.4.3.2 i486-linux-gnu` has been used.

4.1.2 Operative System

Other of the advantages C/C++ offers is portability to other systems. Because this programming language is multi-platform, is possible to compile it in any Unix environment (Linux, BSD, Solaris, Mac, etc...) or Windows environment. For this study, `Ubuntu 8.10 - the Intrepid Ibex` (released in October 2008) has been used.

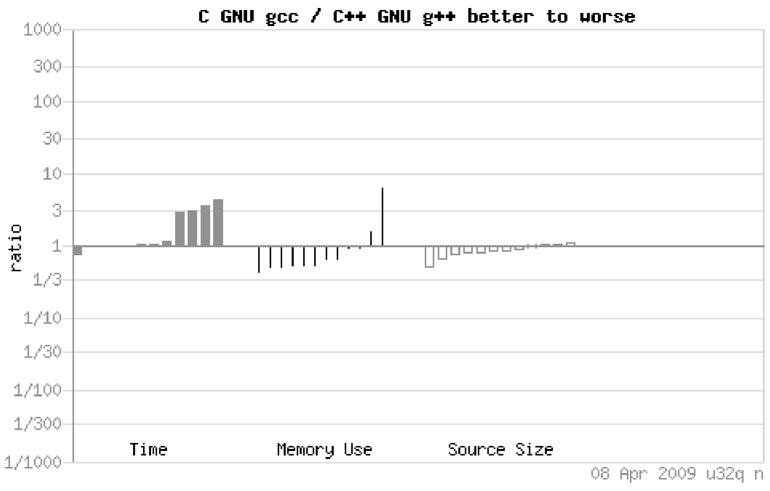


FIGURE 4.1: Comparison between C and C++ performance.

4.1.3 Graphic User Interface Libraries: Qt4

To make the usability easier towards the final user, a graphic interface has been designed in Qt4. The Qt4 libraries were created originally by TrollTech [1]. In June 2008 Nokia acquired Trolltech to enable the acceleration of their cross platform software strategy cross-platform for mobile devices and desktop applications, and to develop its Internet services business. On September 2008 Nokia renamed Trolltech to Qt Software. In fact, Qt libraries are a very good solution to develop standard interfaces. Qt is a cross-platform application framework. Using Qt, it is possible to develop applications and user platform interfaces once, and deploy them across many desktop and embedded operating systems without rewriting the source code. Some of the systems that base part of the code in Qt are consuming electronics (mobile handsets, GPS navigators), manufacturing (part of the Siemens manufacturing assembly line), medical (digital radiology), entertainment (film production) or transportation (mobility systems). The original idea of this project was to implement the algorithms in a standard laptop/PC but it is interesting to mention that could be a feasible option to re-pack the application and install it in embedded systems, PDAs or hand-helds to make the idea portable.

Qt platform is governed by the GNU General Public License version 2 and version 3 so can be used freely as long as it is not used with commercial purposes. There are two possible licenses for Qt4, the *open source* and *commercial.* For the followed targets on this project, open source version has been selected.

[1]More info: `http://www.qtsoftware.com`

4.1.4 Computer Vision Libraries

After the study of the Computer Vision basic algorithms and digital image processing fundamentals, one of the first issues to deal with is the complexity of the functions to be used to create a useful application. Because Computer Vision is on every robotic engineer's lips, several libraries of public distribution have been released amongst which two of them stand out. One of the objectives of the project was to investigate which of them are available and select the most adequate. As it is mentioned before, libraries must be written in C++ in order to be included in the main program.

4.1.4.1 CImg: A Templated Image Processing Library in C++

The CImg Library is a free C++ tool-kit providing simple classes and functions to load, save process and display images in C++ code [2]. It consists only of a single header file CImg.h that must be included in the program source. It contains useful image processing algorithms for loading/saving, resizing/rotating, filtering, object drawing (text, lines, faces, ellipses, ...). Images are instanced by a class able to represent images up to 4-dimension wide (from 1-D scalar signals to 3-D volumes of vector-valued pixels), with template pixel types. One of the big advantages of this libraries is that it depends on a minimal number of libraries: it is easy to compile it with only standard C libraries. There is no need for exotic libraries and complex dependencies.

4.1.4.2 OpenCV: Open Source Computer Vision Library

The most popular free library mainly aimed at real time Computer Vision [3]. It includes a complete list of resources and manuals to interact fully with the libraries from the main application, enhancing the possibilities and the expectations of the project. This point and its easy merger with the graphical libraries Qt4 mentioned before made OpenCV the best choice to implement the application for this project (the extension of this library exceeds the limits of this project). Some example areas covered by OpenCV would be Human – Computer Interaction (HCI), object identification, segmentation and recognition, face recognition, gesture recognition, motion tracking, motion understanding, Structure From Motion (SFM) and mobile robotics.

[2]More info at: `http://cimg.sourceforge.net/`

[3]more info at: `http://sourceforge.net/projects/opencvlibrary/`

4.1.4.3 Selection of the vision libraries

To make the application easy to use and more standard, OpenCV library was chosen for the project. There are hundreds of websites with examples and FAQ's to understand how it works, its functions and problems. Therefore, this option made the integration of Computer Vision algorithms and warping algorithms easy.

4.2 Schema of the application

Because the entire software has been developed in C++, is straightforward to describe the classes involved in the application and their distribution but the relationships and connections are in some cases complex to be represented.

4.2.1 Schema of the application

To describe it later on, a list with the classes with a briefly explanation about the purpose of each class is presented in Table 4.2.1

Name of the Class	Description
AutomaticWidget.cpp	Interface for the automatic average generator window.
CameraCapture.cpp	Capture frames and look for faces inside it.
Detector.cpp	HaarCascades engine.
EigenFaces.cpp	EigenFaces engine.
GraphWidget.cpp	Manual average engine. Nodes, edges and warped images manager.
Main.cpp	Program launcher.
MainWindow.cpp	Main application interface, tool menus and status information handler.
Mixer.cpp	Manual average merger engine.
MixerAut.cpp	Automatic average merger engine.
Node.cpp	Representation of each node included in GraphWidget designed for manual mode.
NodeAut.cpp	Representation of each node included in GraphWidget designed for automatic mode.
SearchWidget.cpp	Interface for the search face window.
StoneFaceWidget.cpp	Automatic average engine.Manager of all the automatic features.
WelcomeWidget.cpp	Interface for the initial menu.

Table 4.1: Description of the classes programmed for the application

In Figure 4.2 there is a representation a global view of the life cycle of the application. In short, there upper-level classes are responsible for the user interface (creation of

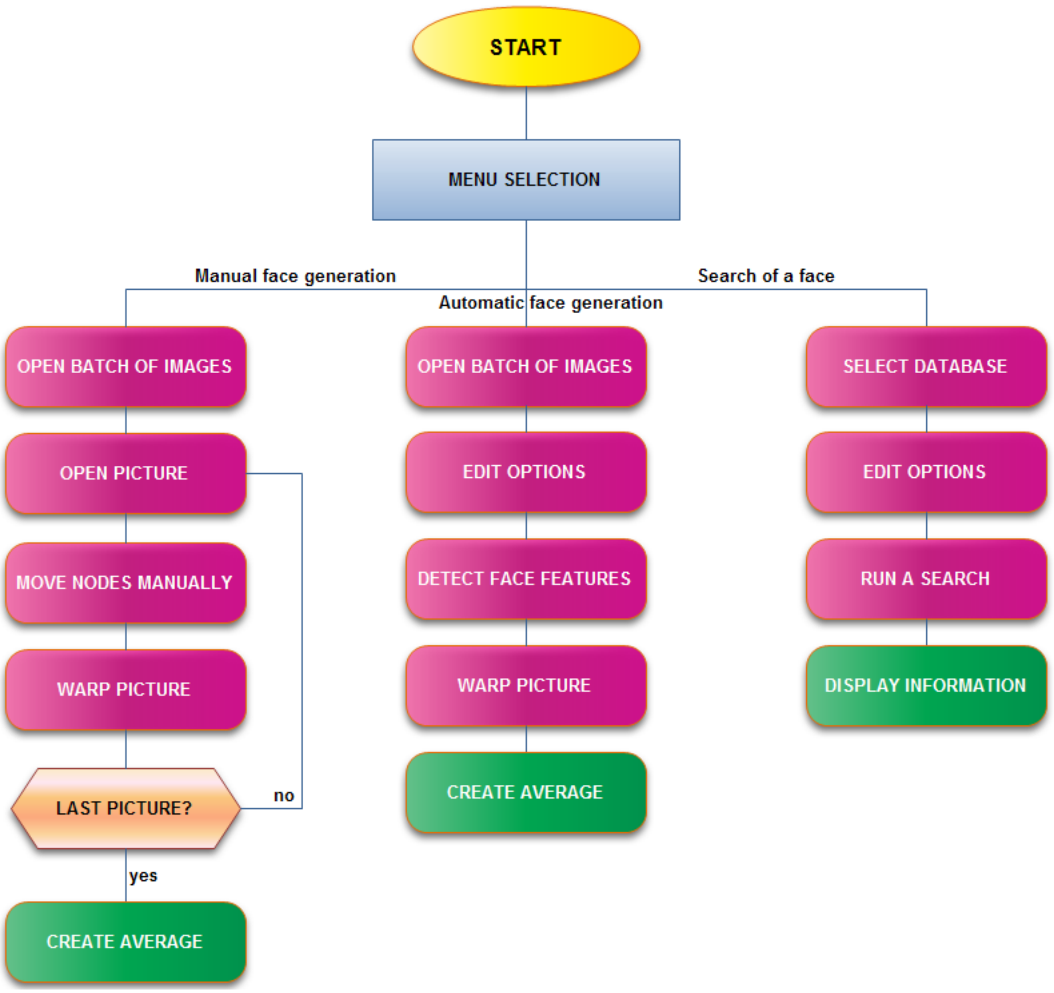


FIGURE 4.2: Schema with the classes and the links between them.

menus, buttons, list-views, user-dialogues or display alerts) while the bottom-level classes manage the mathematical functions (open images, read them, warp them, compute meshes, search for faces, merge files, ...). A complete schema with the methods and classes connections can be found in Appendix C

4.2.2 Interface objects and custom objects

The connection between the interface objects and the internal classes is done trough *slots* and *signals*. The function that relate slots and signals is called *connect* and it is defined as:

BOOL QObject::connect (const QObject *** sender**, const char * *signal*, const QObject *** receiver**, const char * *method*, Qt::ConnectionType type = Qt::AutoConnection)

An example extracted from [32] connects a label with a scrollbar. In this case, when the user interacts with the scrollBar object, it will emit a signal called valueChanged(int) that has an argument `int` that contains the actual value of the scrollBar. The label will display whatever is set in the only argument `int` of its function setNum(int). Because this function is linked to the *valueChanged(int)*, the label will represent directly the value of the scrollBar.

```
QLabel *label = new QLabel;
QScrollBar *scrollBar = new QScrollBar;
QObject::connect( scrollBar, SIGNAL(valueChanged(int)),
                  label,     SLOT(setNum(int)));
```

With this method, it is possible to capture all the events or actions the user perform in the application (clicks, drag drop, keys pressed, etcetera...) and treat them independently.

4.3 Description of the application

The usage of the application is very simple. Once the user execute it, a menu will appear offering three options in form of big icons [4]. Depending on the selection, three different windows could appear.

4.3.1 Manual average

4.3.1.1 Interface

The first option open a full screen window with menu bars, toolbars and a status bar ready to open a batch of images and let the user to decide the node's position in order to generate a face triangular mesh and afterwards warp the image.

It is possible to customize the location and size of the menus in order to make a friendly user environment. Furthermore, is possible to hide the right head with the help nodes to save space and also to show/hide the mesh to make the image more clear (for full information see *User's manual* in Appendix D). The left toolbar is designed to let the user to move the nodes grouped by face features (eyebrows, eyes, mouth, nose, etcetera...) or individually. Furthermore, the mesh has been coloured in order to distinguish better the different groups of nodes.

[4]All the resources of the application have been downloaded from `http://www.crystalxp.net`(last visited April. 10, 2009) and are free to use and distribute by the GPL

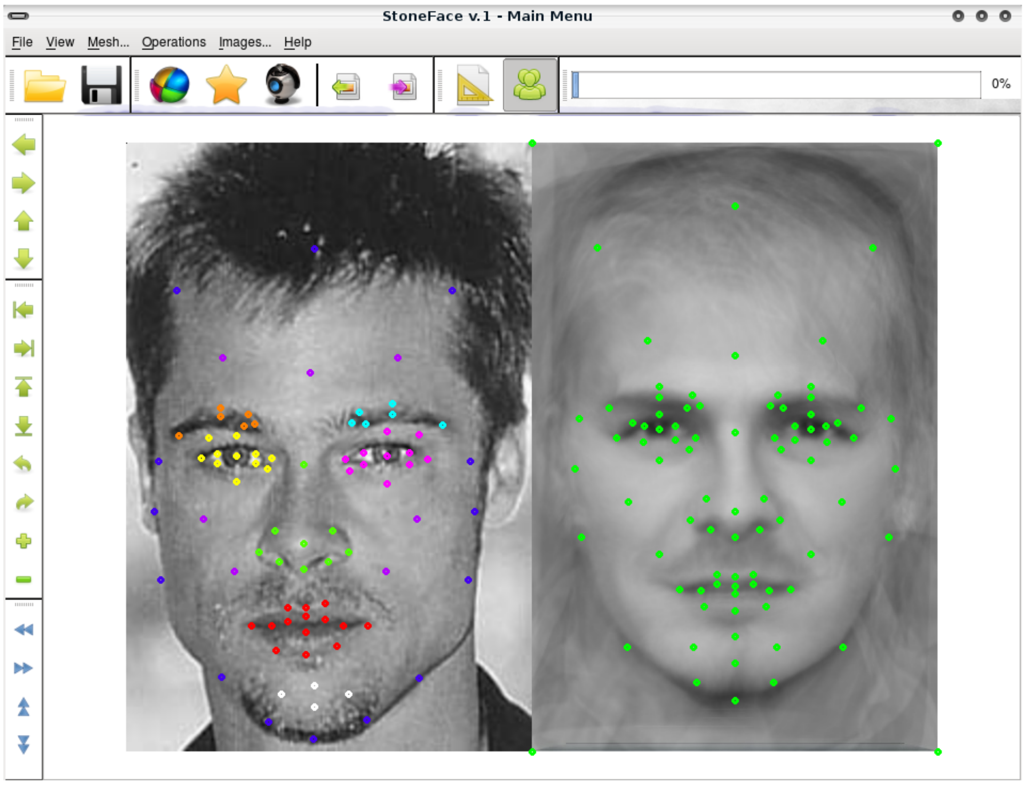


FIGURE 4.3: Screenshot. Manual average face creator.

4.3.1.2 Manual mesh

The mesh is made up of 85 nodes distributed along the whole face forming a triangular mesh. It was designed in order to enhance some important face features like eyes or mouth [5](for full information see Appendix E). That is the reason why there are more nodes around this areas. The mesh can be modified easily creating new results highlighting other points/features. The level of detail of the deformation is directly linked with the number of nodes of the mesh to be warped. As long as we increase the number of nodes, the deformation will be more accurate and therefore more complex to perform for the machine. In the following Figure 4.5 the mesh is displayed. The results obtained of applying this mesh can be read later on Chapter 4.

4.3.2 Automatic average

4.3.2.1 Interface

The second option open a window with three main parts (see Figure D.6). The top one ask the user to introduce the batch of images to analyse. Afterwards, some option

[5]Originally designed by Dr. Rob Jeckins. Psychology Department. University of Glasgow (2009)

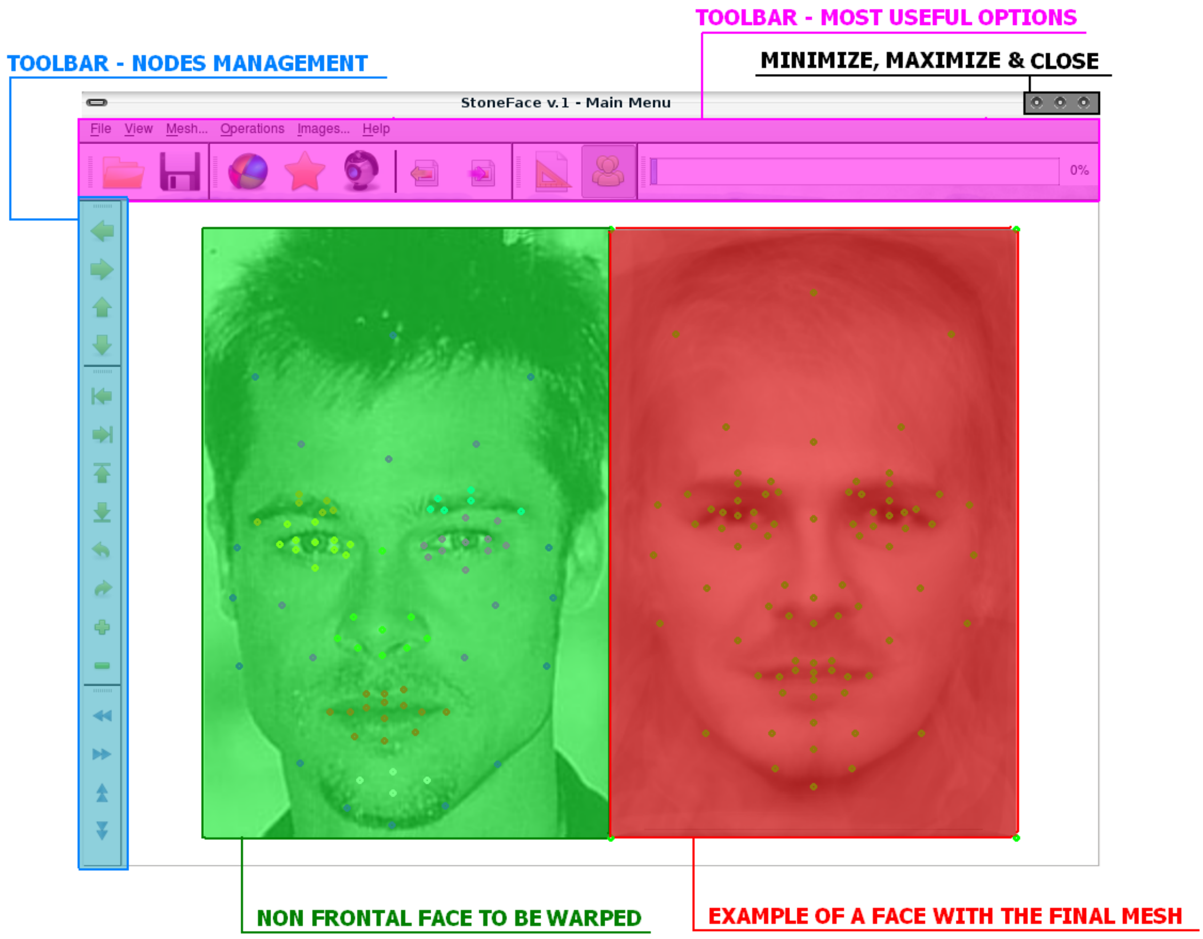

TOOLBAR - MOST USEFUL OPTIONS
TOOLBAR - NODES MANAGEMENT
MINIMIZE, MAXIMIZE & CLOSE
StoneFace v.1 - Main Menu
File View Mesh... Operations Images... Help
0%
NON FRONTAL FACE TO BE WARPED
EXAMPLE OF A FACE WITH THE FINAL MESH

FIGURE 4.4: Screenshot. Highlight of the different sections of the manual form: toolbars, menus and working area.

about the output image can be defined like the colour space (colour or grey scale) or the final size. Finally pressing *Create automatic average...* button, the application will start to open each one of the images and will use Haar Cascades to detect the required face features. It is important to notice that if any of the 5 things searched for any face (face, left eye, right eye, mouth, nose) is not found, this face will be automatically rejected. The reason for that is because the application cannot create a mesh if it unknowns the position of any of the nodes involved (consequences of this decision will be discussed later on Chapter 4).

4.3.2.2 Automatic mesh

Contrary to the previous mesh, the automatic mesh was designed to be less complex. There were two different versions of this mesh. The first one was developed with 34 nodes and the second one was just 12 nodes. After some experiments it was decided that the results did not vary so much between both meshes due to location of points like

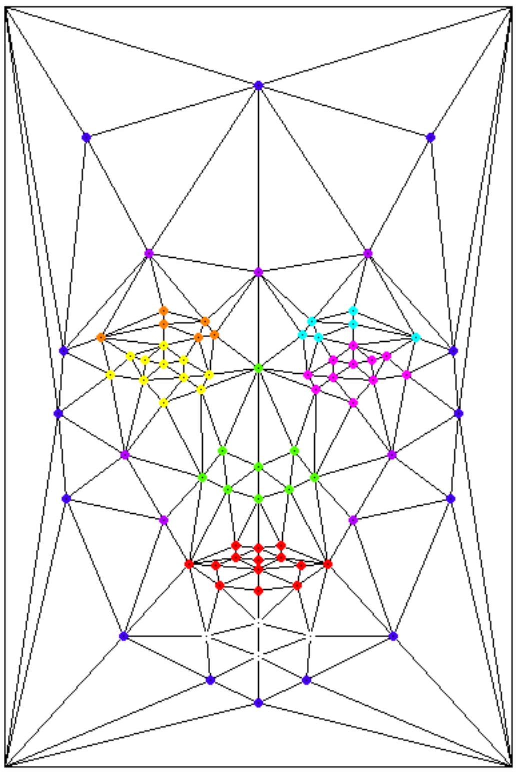
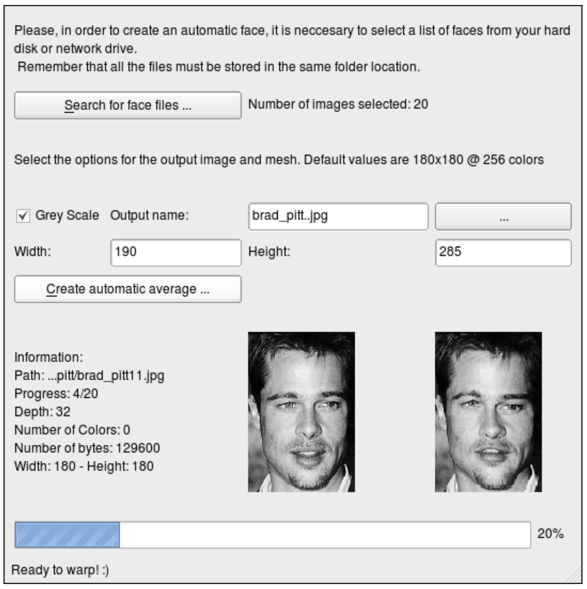


FIGURE 4.5: Triangular mesh designed for the creation of a manual warped image of a face enhancing key features like eyes and mouth.



Please, in order to create an automatic face, it is neccesary to select a list of faces from your hard disk or network drive.
Remember that all the files must be stored in the same folder location.

Number of images selected: 20

Select the options for the output image and mesh. Default values are 180x180 @ 256 colors

☑ Grey Scale Output name: brad_pitt..jpg

Width: 190 Height: 285

Information:
Path: ...pitt/brad_pitt11.jpg
Progress: 4/20
Depth: 32
Number of Colors: 0
Number of bytes: 129600
Width: 180 - Height: 180

20%

Ready to warp! :)

FIGURE 4.6: Screenshot. Automatic average face creator.

eyebrows or chin were poorly estimated in the 34-nodes mesh. As a consequence, the 12-nodes mesh has been implemented for the experiments. Both meshes can be seen in Figure 4.7

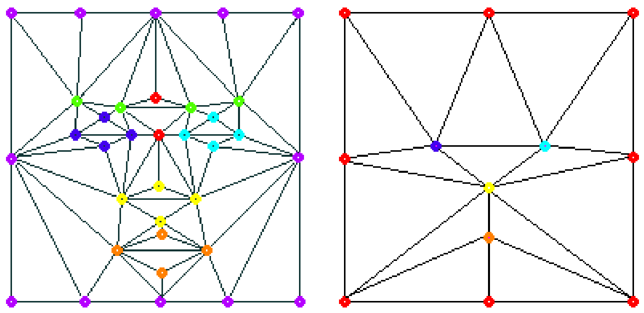


FIGURE 4.7: Triangular meshes designed for the automatic average face generator. Left:34-nodes mesh. Right: 12-nodes mesh.

4.4 Warp algorithm

One of the top features of the project is the warp algorithm implemented. After the study of the most popular ways to deform images in Chapter 3, two of them where implemented. Afterwards, they were compared to decide who was the optimal decision.

4.4.1 Implementation of transformation with multiple pairs of lines

The algorithm was implemented following the steps described in [16] also explained previously. As discussed in [17], the main advantage of this method is that gives the user a high level of control over the process but it is necessary to select corresponding feature lines, being more oriented to morphing rather than warping. Everything that is specified is moved exactly as the animator wants it to move, and everything else is blended smoothly based on those positions. There are two main problems to highlight: [17].

- **Speed** All segments have to be referenced for every pixel. The number of operations is proportional to the number of lines times the number of pixels, making the procedure consumes lots of computer resources.
- **Quality** Not always the result is as expected. The apparition of *ghosts* [15] or undesirable artefacts can be produced.

The results of the implementation of this algorithm are presented on the next Figure 4.8. It is easy to notice that the deformations are inaccurate in some cases and the textures are poorly warped as a consequence of the previous problems (deformations are not completely smooth because the definition of the lines is complex).

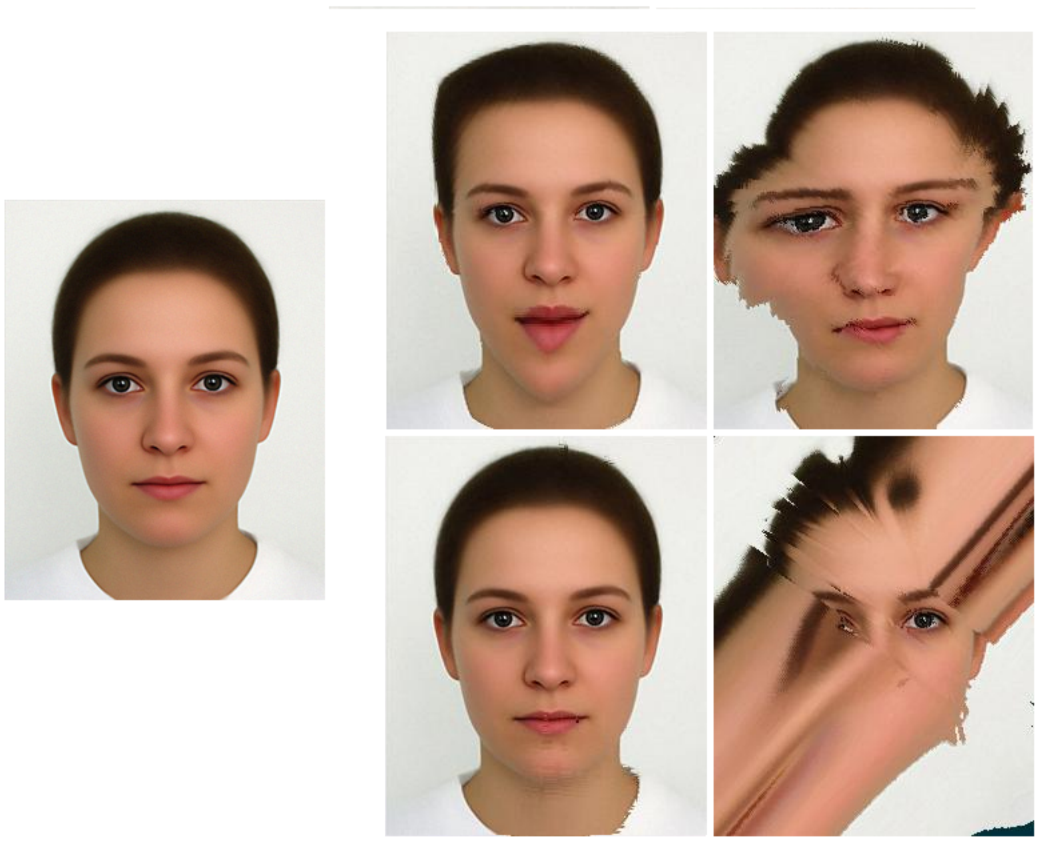


FIGURE 4.8: Results after the implementation of the algorithm transformation with multiple pairs of lines.

4.4.2 Implementation of triangular mesh warping

After the implementation of the previous method, a different solution was need and that is why this algorithm was written. The computation cost is close to mesh warping and the complexity is lower. It is not necessary to use any mesh interpolation like Cat–Mull Algorithm (full code in Appendix A) saving large computational cost.

The steps to follow are not complex. Taking advance of the properties of barycentric coordinates in triangles, the algorithm will take for all the pixels in the destination following the next steps:

Where the interpolation implemented to determine the location of the pixel in the original image is based on the four areas created when x or y does not match with any pixel position. A simple weight is done between these areas instead of using the nearest

Algorithm 4.1: Mesh warping procedure

```
Data: original mesh $M^{orig}$, warped mesh $M^{warp}$, original image $I^{orig}(W \times H)$
Result: warped image $I^{warp}(W \times H)$
1 foreach pixel $P^{warp}$ in $I^{warp}(W \times H)$ do
2     find vertices $A^{warp}, B^{warp}, C^{warp}$ of the triangle $T^{warp}$ in $M^{warp}$ where $P^{warp}$ is
      contained ;
3     calculate $\alpha, \beta$ and $\gamma$ of $T^{warp}$ with $A^{warp}, B^{warp}, C^{warp}$;
4     find vertices $A^{orig}, B^{orig}, C^{orig}$ of the triangle $T^{orig}$ with the relation
      $T^{orig} \leftrightarrow T^{warp}$;
5     calculate the coordinates of the pixel $P^{orig}$ in $I^{orig}$ as
6      $x = \alpha \cdot A_x^{orig} + \beta \cdot B_x^{orig} + \gamma \cdot C_x^{orig}$;
7      $y = \alpha \cdot A_y^{orig} + \beta \cdot B_y^{orig} + \gamma \cdot C_y^{orig}$;
8     copy the value of $P^{orig}$ in $P^{warp}$;
```

neighbour interpolation. To obtain the value of the pixel, the four surrounding pixels share out their values proportionally, resulting:

$$P^{orig} = P_{i,j} \cdot (X_{right} \cdot Y_{down}) + P_{i+1,j} \cdot (X_{left} \cdot Y_{down}) + P_{i,j+1} \cdot (X_{right} \cdot Y_{up}) + P_{i+1,j+1} \cdot (X_{right} \cdot Y_{down}) \tag{4.1}$$

Figure 4.9 shows the names of the variables involved to facilitate the understanding of the method applied.

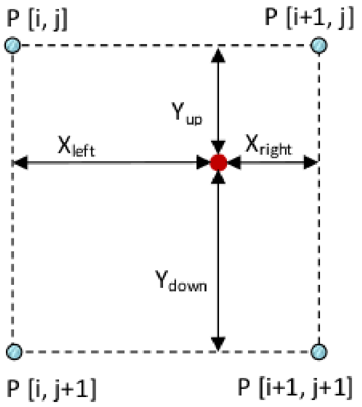


FIGURE 4.9: Interpolation of a pixel over a new mesh.

Some examples of the results given by this algorithm can be shown in Figure 4.10 where deformations are smoother and clear. The quality of the morphing for this method depends directly on the level of detail and the size of the triangles. As long as the quantity of triangles is increased, the deformation of each triangle is lower and therefore the total transformation is less "sharp". Other of the advantages for this method is that the mesh is straightforward to deal with (it is much easier to imagine a mesh than a set of lines representing outlines of the face) allowing the user to change it comfortably, intuitively and just modifying one file with the coordinates. Therefore, for the final version of the project, the triangular mesh warping method was selected.

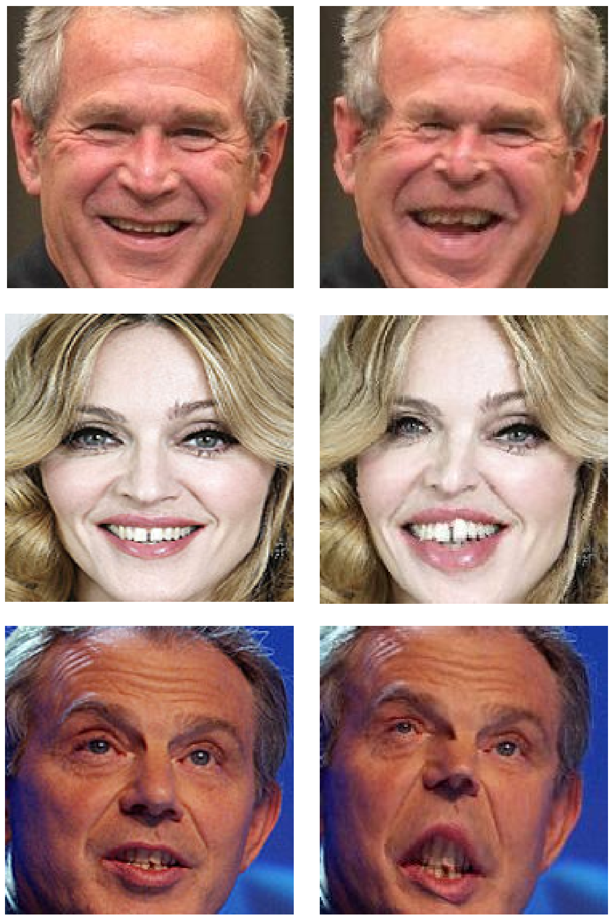


FIGURE 4.10: Results after the implementation of the algorithm triangular mesh warping. George Bush, Madonna and Tony Blair

4.5 Looking for face features

4.5.1 Cascade files management

Exactly as it was discussed previously, the pattern recognition was performed using OpenCV libraries adapted to the project environment. The classes in charge of this task are GraphWidget (manual performance), Detector (Haar cascades management) and StoneFaceWidget (automatic performance). To do that, it is essential to build a cascade's file with a collection of positive and negative samples of frontal faces, left eyes, right eyes, mouths and noses. Because it takes a lot of time to prepare this files, they have been downloaded from a specialized open source website [6] were the files are already prepared and checked to be the best updated approach[34]. A list of the files included in the project can be checked in the next Table 4.5.1

Description	Author(s)	Vers.	Filename
Frontal Face stump 24x24	Rainer Lienhart	1.0	frontalface_alt_tree.xml
Right Eye 18x12	Modesto C. Santana	1.0	ojoD.xml REye18x12.zip
Left Eye 18x12	Modesto C. Santana	1.0	ojoI.xml LEye18x12.zip
Frontal eyes 35x16	Yusuf Bediz	1.0	frontalEyes35x16XML.zip
Mouth 25x15	Modesto C. Santana	1.0	Mouth.xml
Nose 25x15	Modesto C. Santana	1.0	Nose.xml Nose25x15.zip

Table 4.2: Haar-cascades files used to implement the application

4.5.2 Implementation of the Haar Cascades algorithm

The complete code written to deal with Haar Cascades (extracted partially from [35], [36])is very extensive to be copied directly here but the main points to understand are:

1. Create an image using the `cvCreateImage` function.
2. Empty the image.
3. Create a Detector object passing as parameter the image created already.
4. Create the cascade taking into account the feature to be recognized(mouth, nose, eyes ...) using the expression `cascade = (CvHaarClassifierCascade*)cvLoad (cascadePathName, 0, 0, 0 );`
5. Create a sequence of recognized images using the expression `CvSeq* faces = cvHaarDetectObjects( image, cascade, storage,1.1, 5, CV_HAAR_DO_CANNY_PRUNING);` where the function `cvHaarDetectObjects` looks for rectangular regions in the created image that are likely to contain objects the cascade has been trained for and returns those regions as a sequence of rectangles. The function scans the image several times at different scales[7].
6. Create a new rectangle to pick up the coordinates using the expression `CvRect* r = (CvRect*)cvGetSeqElem( faces, i);` for each region found in the image.
7. Extract the image embedded in this rectangle using the previous data.
8. Set the mesh points applying the coordinates given.

[6]Website: `http://alereimondo.no-ip.org/OpenCV/34` last visited April. 10, 2009

[7]More info can be found in `http://www710.univ-lyon1.fr/~bouakaz/OpenCV-0.9.5/docs/ref/OpenCVRef_Experimental.htm`

Some examples of the Haar Cascades results are displayed in the next figures. In Figure 4.11 an example of a clear frontal face is detected including all the features: eyes, node and mouth. In Figure 4.12 an example of a frontal face is detected. This time the subject wears glasses but both eyes have been found. In Figure 4.13 an example of multiple people face detection is displayed.

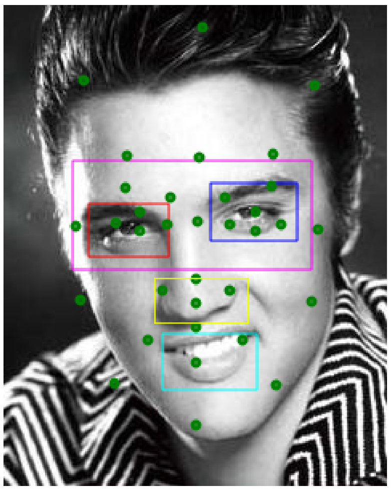


Figure 4.11: Haar Cascade examples. Face location, face features location without occlusions.

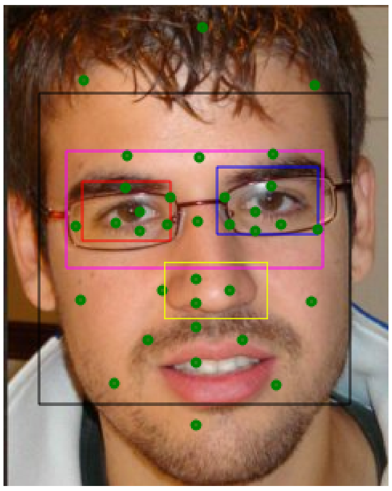


Figure 4.12: Haar Cascade examples. Face location, face features location with glasses.

4.6 Face recognition

4.6.1 Eigenfaces using OpenCV

The last part of the project consist on the development of a simple face recognition module to demonstrate if the results are or not valid. The OpenCV libraries include a

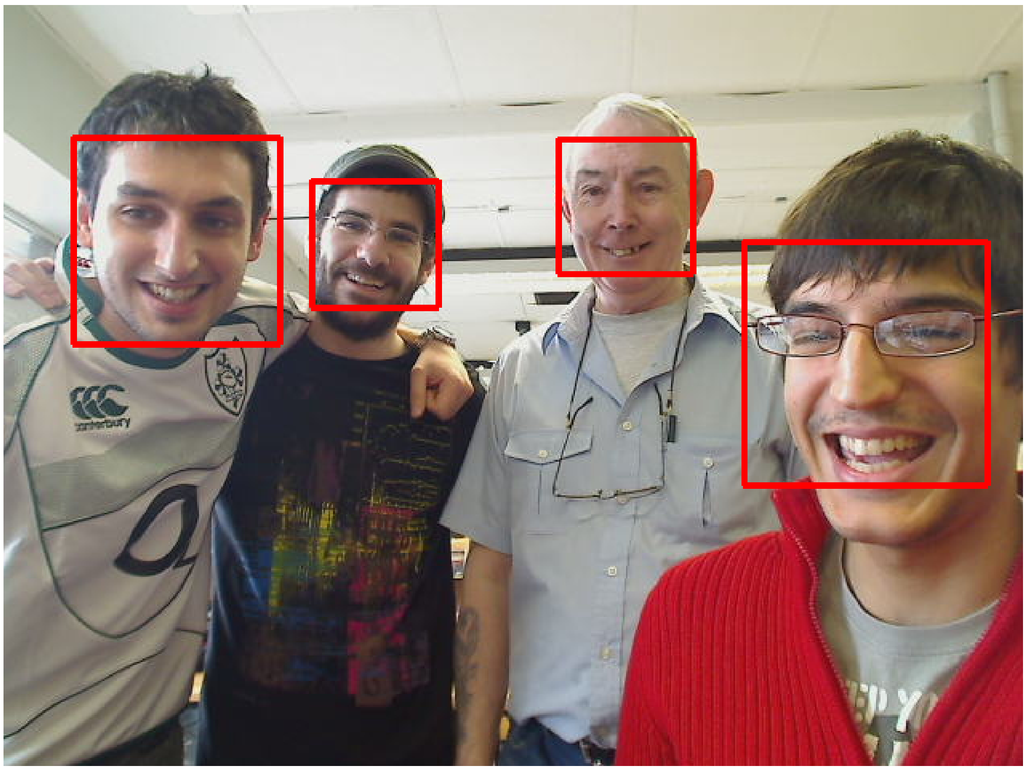


FIGURE 4.13: Haar Cascade examples. Face location inside a multiple faces in the laboratory.

module dedicated to PCA, that means, it is ready for eigenfaces applications. The code used for this part was extracted based on a previous project [8] with several modifications in order to be compatible with the actual project application.

The steps followed to create the module are:

1. Creation of a Database with all the eigenfaces.
2. Application of PCA to the Database to reduce the information
3. Load training information of the Database created.
4. Project the test image into the PCA subspace
5. Find the closest distance between the subspace and the projected face listings

[8]Check `http://www.codeproject.com/KB/cpp/TrackEye.aspx`

Chapter 5

Results and Discussion

In this chapter, some studies and experiments will be discussed to understand if the average of a face increase the accuracy measurement of a face recognition software. Furthermore, several comparisons have been run to determine if an automatic average face can reach the *quality* and *recognition rate* of a manual average face.

5.1 Database

To apply the algorithms implemented during the previous episodes, a batch of **500** male celebrity images has been selected. All of them have been picked up under several light conditions, different periods of their lives and variant face gestures. All the images are 190x190 pixels in `JPEG` format grey-scale and contain busts. They have been extracted from public websites and were provided to this project by Dr. Rob Jeckins [1]. Some of this samples can be displayed in Figure 5.1

5.2 Experiments realized

Several experiments have been done to demonstrate that the average face of a person is better than any random single face by itself. Begining experiments compare the recognition rates of manual average faces versus automatic average faces to understand the level of similarity between them. Afterwards, some experiments compare the recognition rates between automatic average faces and random single faces. The generated faces are attached in Appendix F and some of them are also displayed in Figure 5.2. It is possible

[1]Psychology Department. University of Glasgow (2009)

Figure 5.1: Example of some single faces introduced in the face database to do the experiments.

to difference between automatic and manual output images and, in conjunction with the following studies, the results achieved with this project. Finally, time comparisons and the evolution of average faces will be done but before starting to compare the warped images, a simple study is done between a *warped average face* and a *non-warped average face*, to demonstrate that the deformation of the images to a frontal position before making the average operation is one of the key points of this investigation.

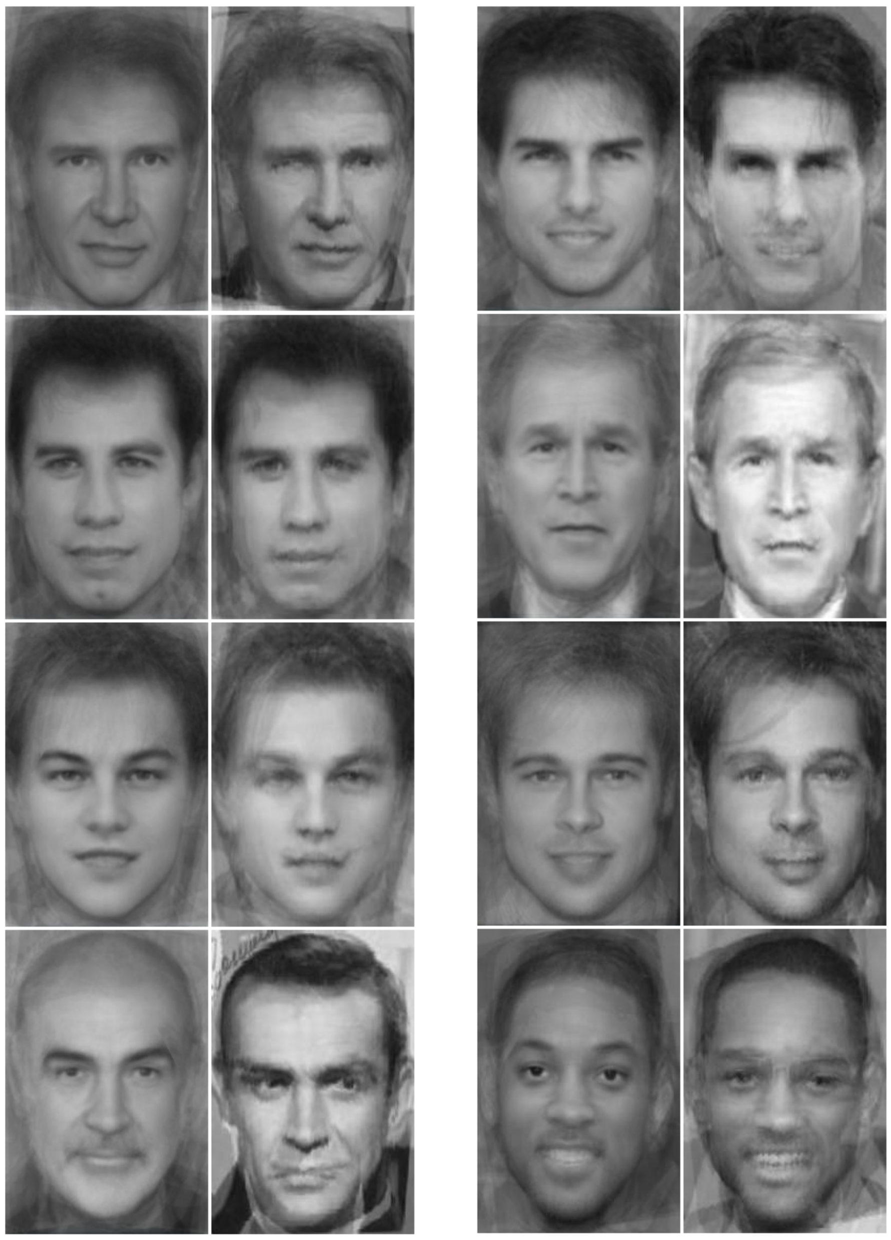


FIGURE 5.2: Average images generated by the application. Left images are manual average faces and right images are automatic average faces.

5.2.1 The importance of warping the images

For the first experiment, a group of non-warped faces has been created and a face recognition algorithm has been applied to them (see Figure 5.3 for some samples and Appendix F for the complete list of faces). To make the experiment complete, the number of faces used to generate the non-warped average is increased to understand the behaviour of the algorithm.

Faces used to create the average	Faces recognised	Percentage
5 faces	9/25	36.0%
10 faces	7/25	28.0%
15 faces	6/25	24.0%
20 faces	5/25	20.0%

Table 5.1: Results of recognition rate in non-warped faces. The number of images used to generate the average changes

Obviously, as long as the number of faces in increased (Table 5.1), the output generated is more *fuzzy* an less similar to a face and therefore the recognition rate is smaller. This is the reason why the deformation followed in this project enhance the chances for a face to be recognized.

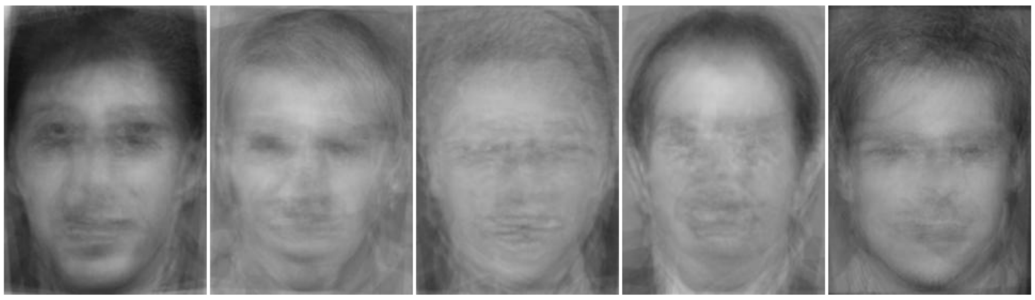


Figure 5.3: Example of some non warped faces.

Those results represent that the deformation of the images before the creation of an average is essential for the correct subsequent recognition.

5.2.2 Manual and automatic average faces

Comparison between automatic and manual average faces in different experiments to see how distant they are one to the other. For face recognition probes, the widely used Principal Component Analysis(PCA or eigenfaces for recognition [26]) has been applied as the classifier for the identifications. Afterwards, Mahalanobis distance metric is used to calculate the nearest match in the space generated by the eigenfaces.

5.2.2.1 First experiment: Average with included images

The first experiment includes 500 faces (20 images x 25 people) in the database and consist on the next steps:

1. Generate an automatic average faces for each celebrity with the photos included in the database.
2. Generate a manual average faces for each celebrity with the photos included in the database.
3. Run the eigenfaces algorithm for all the average images generated.
4. Compare results between manual and automatic average faces.

The results of the experiment are displayed in Table 5.2 and surprisingly return that the percentage of accuracy of the automatic average face is greater than the manual average. The high accuracy of the results in both cases demonstrates that the average face can be a very good representation for artificial face recognition. On the other hand, the reason why the automatic average seems to be better than the manual one is that some of the images generated automatically include just two or three images of each person because the rest have been rejected due to lack of information (shadows that hide eyes, moustaches, etcetera...) while in the case of manual faces, all of the generated faces have been created from 20 images of each celebrity.

Experiment	Faces recognised	Percentage
Manual Face	403/500	80.6%
Automatic Face	481/500	**96.2%**

Table 5.2: Manual average face and automatic average face recognition rates for the first experiment

5.2.2.2 Second experiment: Average with not included images

The second experiment includes 375 faces (15 images x 25 people) in the database and consist on the next steps:

1. Generate an automatic average faces for each celebrity with the remaining 5 photos **not included** in the database.
2. Generate a manual average faces for each celebrity with the remaining 5 photos **not included** in the database.

3. Run the eigenfaces algorithm for all the average images generated.
4. Compare results between manual and automatic average faces.

Experiment	Faces recognised	Percentage
Manual Face	123/375	32.8%
Automatic Face	152/375	**40.53%**

TABLE 5.3: Manual average face and automatic average face recognition rates

Once again, the automatic faces generated are closer to the generated database than the manual faces (Table 5.3). The recognition rate has decreased because the faces to be recognised are not included in the database and therefore the generated face is not a combination of database faces.

5.2.2.3 Third experiment: Database formed with averages. Single faces included

The third and forth experiments correspond to a change in the point of view of the problem. If instead of trying to recognise average faces in a single-face database, the average faces are the ones that create the database, results may change. In this case, the average faces create a subspace of images, reducing substantially the number of images inside the database (×20).The difference between both experiments is that in one the single images are part of the images used to generate the average and in the other one are not included.

The steps followed for the third experiment are:

1. Introduce all the manual averages of the 20 images in the database
2. Try to recognise single images (5) included in the database.
3. Introduce all the automatic averages of the 20 images in the database
4. Try to recognise single images (5) included in the new database.

Experiment	Faces recognised	Percentage
Database of manual faces	63/125	50.4%
Database of automatic Face	50/125	40.0%

TABLE 5.4: Database created with average faces using single included images to be recognised

The results obtained developing this method decrease the recognition rate significantly smaller as shown in Table 5.4.

5.2.2.4 Forth experiment: Database formed with averages. Single faces not included

The steps followed for the fourth experiment are:

1. Introduce all the manual averages of the 20 images in the database
2. Try to recognise single images (5) **not included** in the database.
3. Introduce all the automatic averages of the 20 images in the database
4. Try to recognise single images (5) **not included** in the new database.

The results obtained developing this method decrease the recognition rate drastically as shown in Table 5.5 due to results in previous section and the no including of recognizing faces to create the average.

Experiment	Faces recognised	Percentage
Database of manual faces	10/125	8.0%
Database of automatic Face	13/125	10.4%

Table 5.5: Database created with average faces using single included images to be recognised

5.2.2.5 Fifth experiment: Comparison between manual and automatic average faces

Other interesting aspect to compare both kind of averages (manual and automatic) is studying the correlation between both. That is, see how many automatic faces are recognised in a database of manual faces and vice versa.

Experiment	Faces recognised	Percentage
Database of manual faces	19/25	76.0%
Database of automatic Face	17/25	68.0%

Table 5.6: Correlation between both: manual and automatic faces

The results in previous Table show that there is not very different to create the database in one way or in the other because results are very close. That means, the level of recognition of one batch compared from the other one is high and therefore both groups of images are close (in terms of subspace generated by each one of the batches).

5.2.3 Automatic single faces and average faces

In this second part, a different aspect will be examined: the difference between search a single face or an average face inside a database. For this section, instead of the PCA method used before to recognise the faces, an external application is going to be used. As in [37], the website `www.myheritage.com` will be used. This website has an online application for face recognition as mentioned in Chapter 2 that detects faces inside photos and displayed the closest celebrities with a percentage of similarity.

5.2.3.1 First experiment: Study comparing single faces with automatic averages including non-matched faces

The first experiment compares the submission of a random single face and the submission of an automatic average face. For this, all the non-rejected images used to generate the automatic faces have been submited and search in `www.myheritage.com`. In Figure 5.4 the results are displayed including all the faces that the website failed to recognise. Examining the mentioned Figure, there are 5 average face that where not recognised (George Clooney, Jack Nicholson, Will Smith, Sean Connery and Tom Cruise) while for the rest of the faces the average is more probable to be recognised(12/19) than the average single image (7/19) and just in one submission.

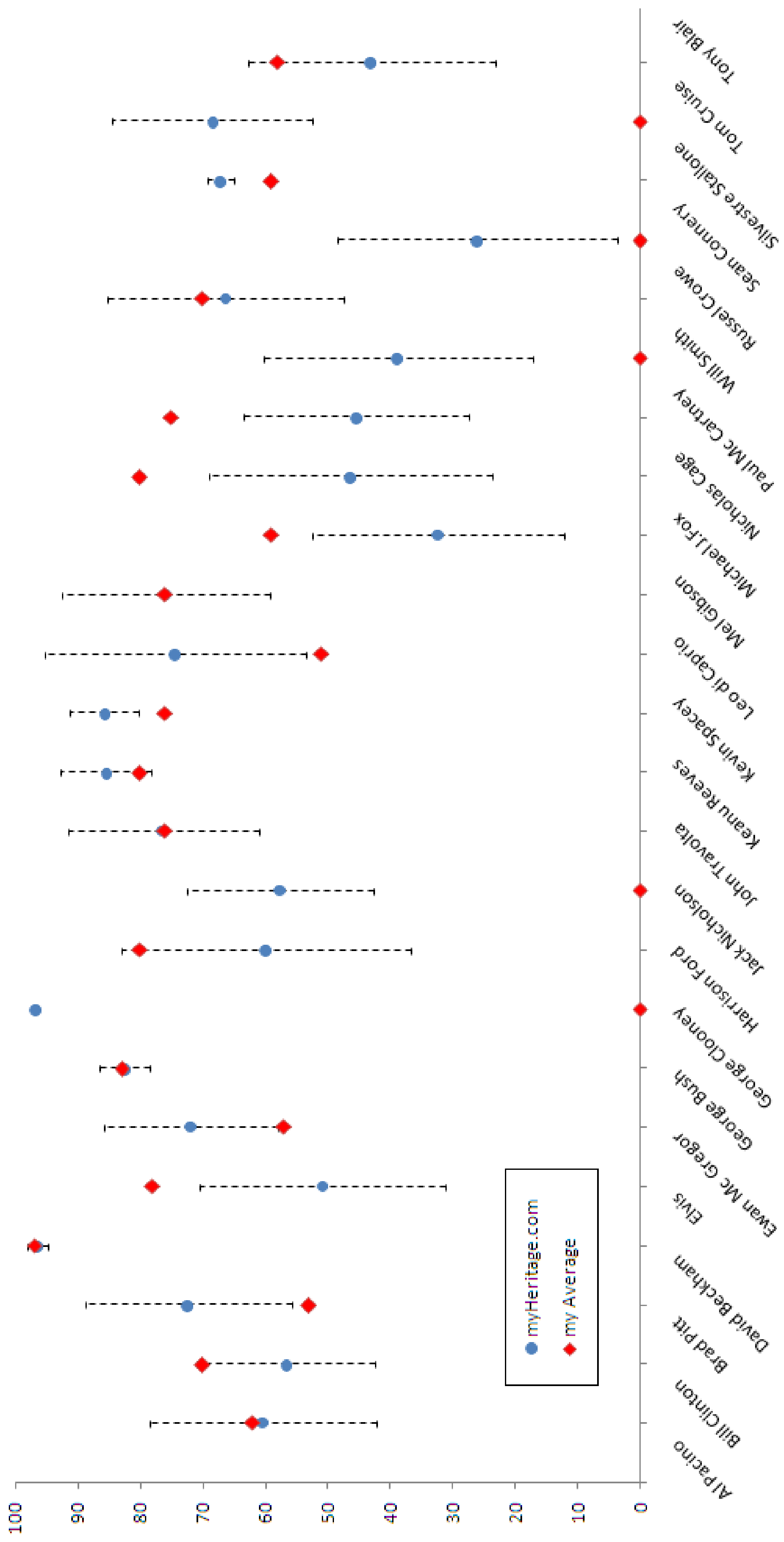


FIGURE 5.4: Results comparing the recognition rate between a single face and the automatic average face (1).

5.2.3.2 Second experiment: Study comparing single faces with automatic averages without non-matched faces

Now if the failed hits are removed, is possible to compare the probability of a face to be recognised being an automatic average or just a random single face.

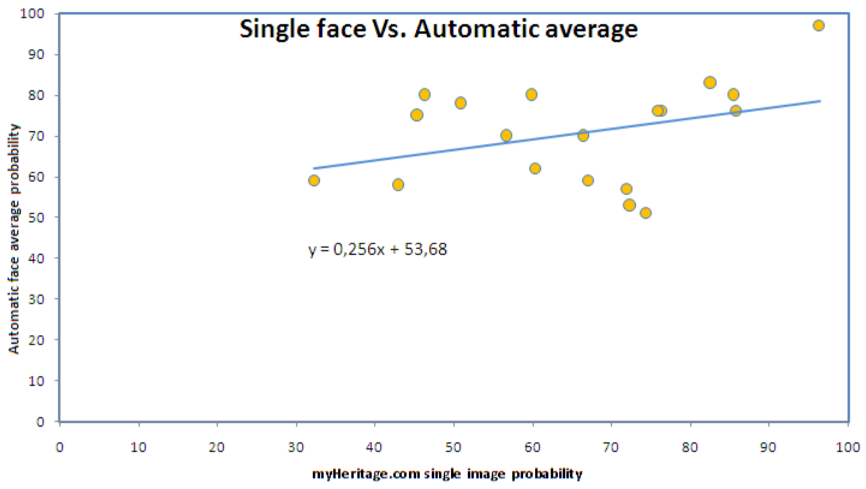


FIGURE 5.5: Results comparing the recognition rate between a single face and the automatic average face (2).

Examining now Figure 5.5, it is concluded that when the probability of a single image to be zero is reached, the probability of the average face is still 53,68%. That makes the average strong and more persistent against changes than a single image (as it was expected).

5.2.3.3 Third experiment: Study comparing single faces with automatic averages without non-matched faces

Now the failed hits are removed to understand how relevant they are for the results.

Looking at Figure 5.6, where the failed hits have been removed for both variables, the average probability is increased slightly because failed hits were more detrimental to the average face than for the individual faces (54,88%).

5.2.3.4 Forth experiment: Single non-frontal faces Vs. automatic average face

To demonstrate that the average face can be a better choice in order to recognise faces, non-frontal faces of each celebrity were introduced in the website to be recognised. The

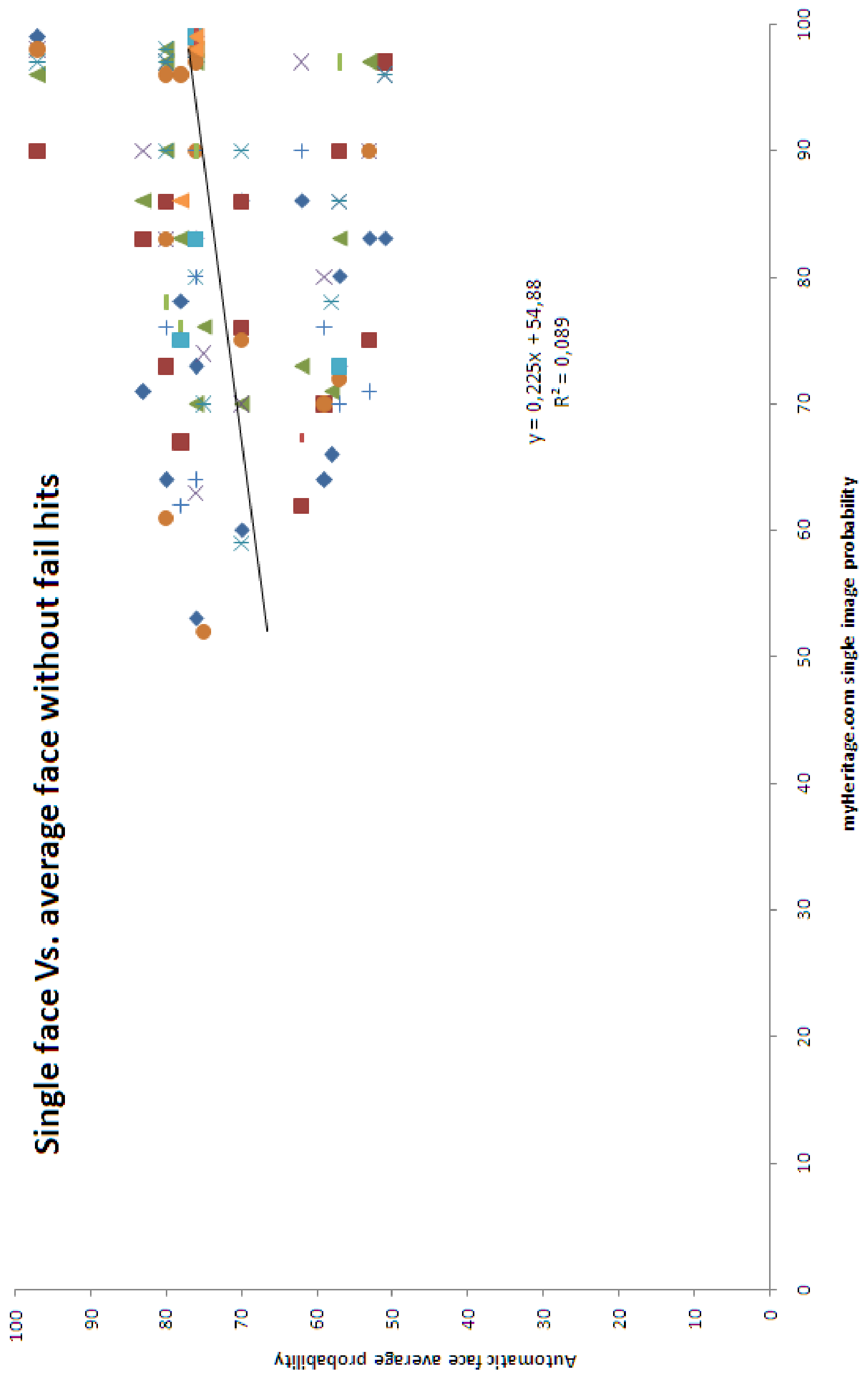


FIGURE 5.6: Results comparing the recognition rate between a single face and the automatic average face (3).

results show that statistically 11 of 24 non-frontal images are not recognized while for average faces the numbers decreases until 4.

The results obtained developing this method decrease the single face recognition rate drastically as shown in Table 5.2.3.4 due to non frontal images are extremelly hard to be recognise for a single images collection (lack of information of one side of the face decreases the overall data about the face).

Experiment	Faces recognised	Rec. rate
Single non-frontal face	13/24	39.29%
Automatic average face	21/24	56.26%

Table 5.7: Single non-frontal faces Vs. automatic average faces

5.2.4 Time comparisons

The main advantage of the automation of the warping algorithm is to save great amount of time. Because the nodes's positioning is hard work and extremely slow, the automatic average face becomes more interesting than the manual average face. To understand the quantities of time expended for both cases, a simple experiment have been done. Time has been measured for both using the `cputime()` function in C/C++ which returns the actual time of the computer when the function is called. The statistical results of this experiment are displayed in Table 5.2.4 for 20 faces.

Experiment	Time expended (sec)
Manual average face creation	2340.46
Automatic average face creation	8.83

Table 5.8: Single non-frontal faces Vs. automatic average faces

It is straightforward to realise that the implementation of the automatic system decreases abruptly the time to generate an average face **265 times!**.

5.2.5 Evolution of average faces

In this section, the behaviour of the recognition rate will be tested increasing the number of images used to create the average image as in [39]. In Figure 5.7 the evolution of the average can be seen. As long as the number of images that compound the average is increased, the recognition rate increases from 63.5% to 96.2% as in Table 5.2.

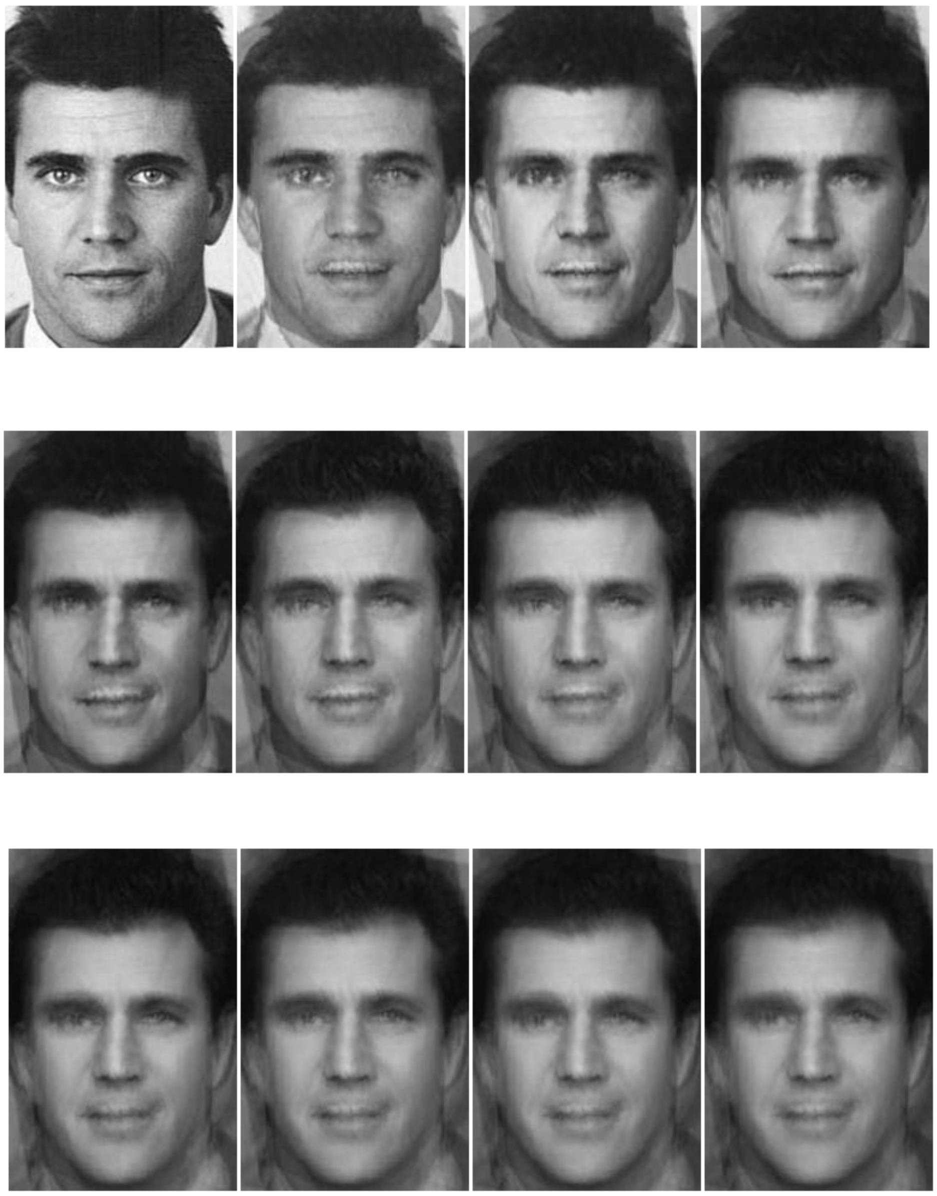


FIGURE 5.7: Results comparing the recognition rate between a single face and the automatic average face (3).

5.2.6 Rejection of faces

The most important problem of using automatic face averaging is the number of images rejected in the process. As mentioned before, in the case that in an image both eyes, nose and mouth are not recognised, this image is removed from the average image. This assumption make manual images contain more information than the automatic faces. For this project, the average of images used for each face was 7.37 / 20 (36.9%).

5.3 Final result

As an overall, the final Figure that resumes the recognition rates obtained during the last experiments is displayed in Figure 5.8. It is evident that automatic average faces is a very good estimator of the complete face of a person.

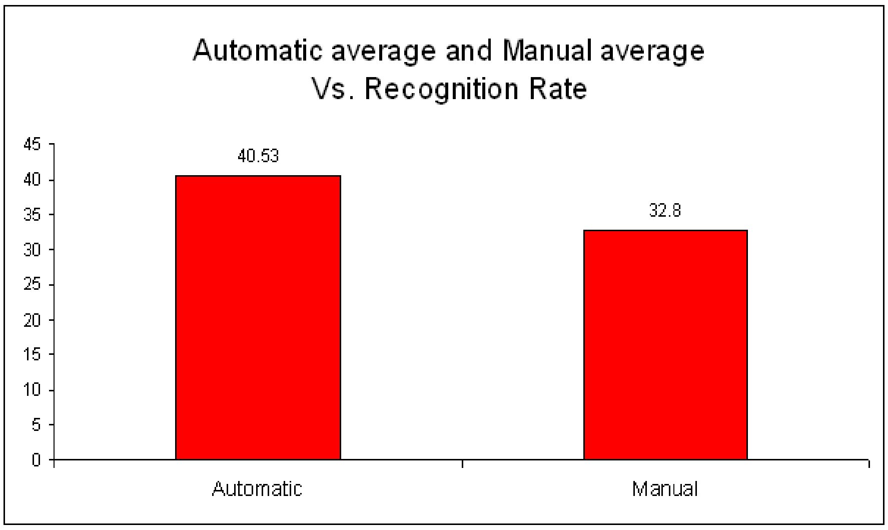


FIGURE 5.8: Results comparing the overall recognition rate

Chapter 6

Conclusions and future works

6.1 Conclusions

In the present work, a new average facial generator has been designed combining triangle-mesh warping and Haar Cascades. A basic face recognition system has been developed using PCA and eigenfaces to determine the quality of the results obtained. To complete the investigation, an external face recognition tool provided by `http://www.myheritage.com` has been used in order to contrast the results.

With this study some important aspects are concluded:

1. The deformation of the faces before generating the average face increases rapidly the recognition rate because of the control of the position of the face features: eyes, mouth and nose.
2. The number of images used to create the average affects radically to the recognition rate. It increases when new images of the individual are added while creating the average .
3. When a face image is non-frontal, has badly light conditions or low resolution the average improves the recognition rate.
4. The time spent for the generation of a manual average face is 265 times the time necessary by the solution presented. Because 40 minutes to generate a manual average face is too long, this innovation makes the idea of automatic average faces to be feasible and useful.

5. The correlation between manual and automatic faces is fairly close, providing both similar results being automatic average even better in several cases.

6.2 Future works

After the development of the application and the complete solution to the original planted problem, some future works can be done. Because the application has been developed in C++, all the posterior ideas should be implemented in that way, avoiding complex parsing functions or new bugs. To improve the application, some new ideas can be added and implemented.

- Increase the number of features of the face for the automatic average face. To do that, new Haar Cascades are required, including eyebrows, chin, etcetera...
- Implement the mesh warping algorithm with the Cat-Mull interpolation attached in Appendix A to check if the recognition ratio increases.
- Add new features to the graphic interface such as parameter options configuration, image format management, configuration registry to save the user preferences, improved image management based on thumbnails, etcetera...
- Optimize the source code to gain speed in hard functions like warping or Haar-like feature's search.
- Implement a web-based application that generate average faces based on captured images via webcam.
- Development of a complete real-time application that controls and generates average faces of each person detected in the camera.

Appendix A

Cat–Mull Algorithm. Mesh interpolation

Cat-Mull algorithm in C

```
/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * catmullRom:
 *
 * Compute a Catmull-Rom spline passing through the len1 points in arrays
 * x1, y1, where y1 = f(x1)
 * len2 positions on the spline are to be computed. Their positions are
 * given in x2. The spline values are stored in y2.
 */
void
catmullRom(x1, y1, len1, x2, y2, len2)
float *x1, *y1, *x2, *y2;
int  len1, len2;
{
 int i, j, dir, j1, j2;
 double x,  dx1, dx2;
 double dx, dy, yd1, yd2, p1, p2, p3;
 double a0y, a1y, a2y, a3y;


 /* find direction of monotonic x1; skip ends */
 if(x1[0] < x1[1]) { /* increasing */
  if(x2[0]<x1[0] || x2[len2-1]>x1[len1-1]) dir=0;
```

```
 else dir = 1;
} else {  /* decreasing */
 if(x2[0]>x1[0] || x2[len2-1]<x1[len1-1]) dir=0;
 else dir = -1;
}
if(dir == 0) {   /* error */
 printf("catmullRom: Output x-coord out of range of input\n");
 return;
}

/* p1 is first endpoint of interval
 * p2 is resampling position
 * p3 is second endpoint of interval
 * j  is input index for current interval
 */

/* force coefficient initialization */
if(dir==1) p3 = x2[0] - 1;
else  p3 = x2[0] + 1;

for(i=0; i<len2; i++) {
 /* check if in new interval */
 p2 = x2[i];
 if((dir==1 && p2>p3) || (dir== -1 && p2<p3)) {
  /* find the interval which contains p2 */
  if(dir) {
   for(j=0; j<len1 && p2>x1[j]; j++);
   if(p2 < x1[j]) j--;
  } else {
   for(j=0; j<len1 && p2<x1[j]; j++);
   if(p2 > x1[j]) j--;
  }

  p1 = x1[j];  /* update 1st endpt */
  p3 = x1[j+1];  /* update 2nd endpt */

  /* clamp indices for endpoint interpolation */
  j1 = MAX(j-1, 0);
  j2 = MIN(j+2, len1-1);
```

```
   /* compute spline coefficients */
   dx  = 1.0 / (p3 - p1);
   dx1 = 1.0 / (p3 - x1[j1]);
   dx2 = 1.0 / (x1[j2] - p1);
   dy  = (y1[j+1] - y1[ j ]) * dx;
   yd1 = (y1[j+1] - y1[ j1]) * dx1;
   yd2 = (y1[j2 ] - y1[ j ]) * dx2;
   a0y =  y1[j];
   a1y =  yd1;
   a2y =  dx *  ( 3*dy - 2*yd1 - yd2);
   a3y =  dx*dx*(-2*dy +   yd1 + yd2);
  }
  /* use Horner's rule to calculate cubic polynomial */
  x = p2 - p1;
  y2[i] = ((a3y*x + a2y)*x + a1y)*x + a0y;
 }
}
```

Appendix B

HSI-to-RGB Conversion

Depending on the sector of the color circle, the formula changes.
For $0^o \leqslant H < 120^o$,

$$R = \frac{I}{\sqrt{3}} \cdot \left(1 + \frac{S \cdot cos(H)}{cos(60^o - H)}\right)$$

$$B = \frac{I(1-S)}{\sqrt{3}}$$

$$G = \sqrt{3} \cdot I - R - B$$

for $120^o \leqslant H < 240^o$,

$$G = \frac{I}{\sqrt{3}} \cdot \left(1 + \frac{S \cdot cos(H - 120^o)}{cos(180^o - H)}\right)$$

$$R = \frac{I(1-S)}{\sqrt{3}}$$

$$B = \sqrt{3} \cdot I - R - B$$

and for $240^o \leqslant H < 360^o$,

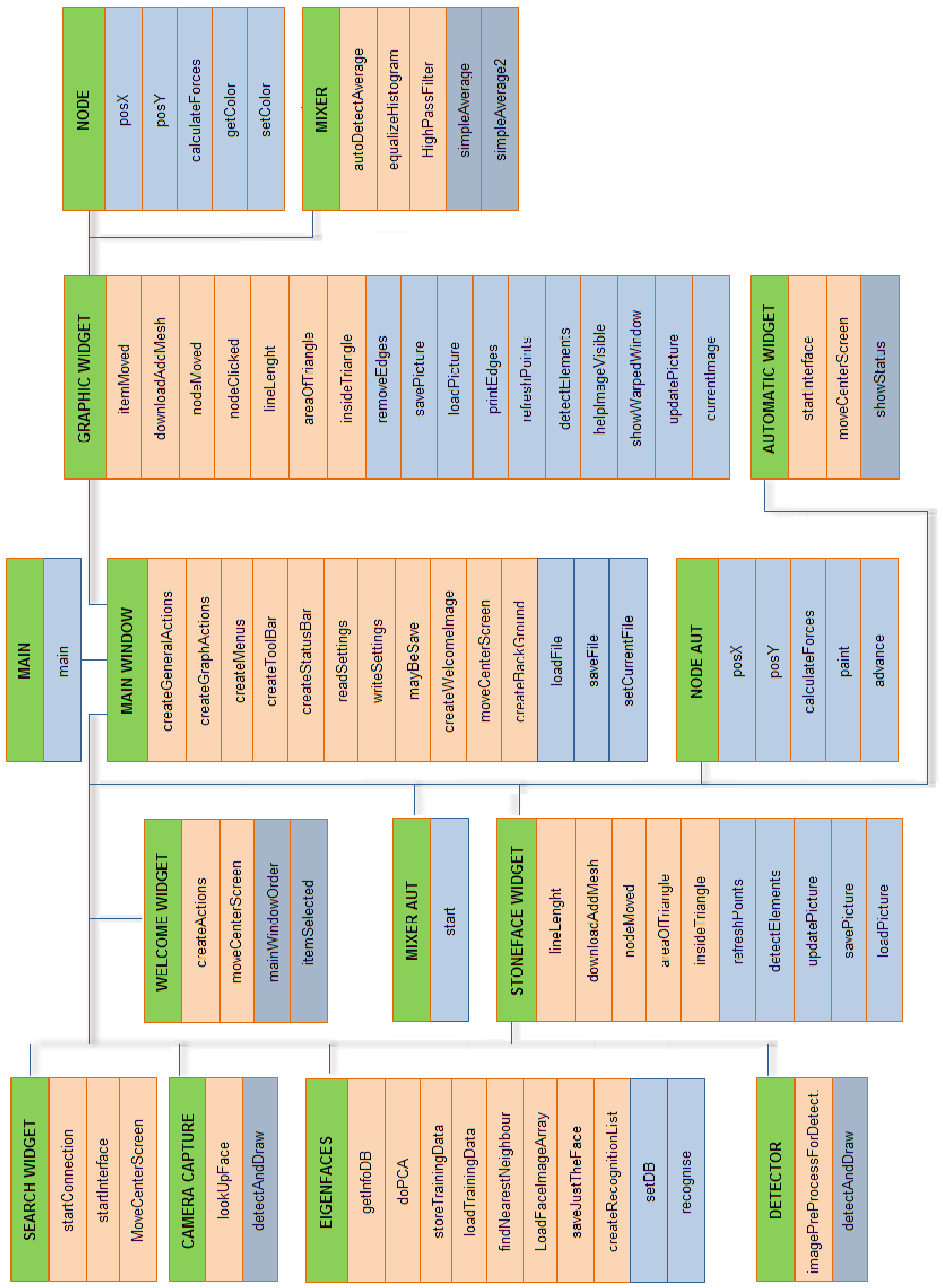
$$B = \frac{I}{\sqrt{3}} \cdot \left(1 + \frac{S \cdot cos(H - 240^o)}{cos(300^o - H)}\right)$$

$$G = \frac{I(1-S)}{\sqrt{3}}$$

$$R = \sqrt{3} \cdot I - R - B$$

Appendix C

Schema of the Application


NODE
posX
posY
calculateForces
getColor
setColor
MIXER
autoDetectAverage
equalizeHistogram
HighPassFilter
simpleAverage
simpleAverage2
GRAPHIC WIDGET
itemMoved
downloadAddMesh
nodeMoved
nodeClicked
lineLenght
areaOfTriangle
insideTriangle
removeEdges
savePicture
loadPicture
printEdges
refreshPoints
detectElements
helpImageVisible
showWarpedWindow
updatePicture
currentImage
AUTOMATIC WIDGET
startInterface
moveCenterScreen
showStatus
MAIN
main
MAIN WINDOW
createGeneralActions
createGraphActions
createMenus
createToolBar
createStatusBar
readSettings
writeSettings
mayBeSave
createWelcomeImage
moveCenterScreen
createBackGround
loadFile
saveFile
setCurrentFile
NODE AUT
posX
posY
calculateForces
paint
advance
WELCOME WIDGET
createActions
moveCenterScreen
mainWindowOrder
itemSelected
MIXER AUT
start
STONEFACE WIDGET
lineLenght
downloadAddMesh
nodeMoved
areaOfTriangle
insideTriangle
refreshPoints
detectElements
updatePicture
savePicture
loadPicture
SEARCH WIDGET
startConnection
startInterface
MoveCenterScreen
CAMERA CAPTURE
lookUpFace
detectAndDraw
EIGENFACES
getInfoDB
doPCA
storeTrainingData
loadTrainingData
findNearestNeighbour
LoadFaceImageArray
saveJustTheFace
createRecognitionList
setDB
recognise
DETECTOR
imagePreProcessForDetect.
detectAndDraw

Appendix D

User's manual

D.1 Introduction to the application

StoneFace will allow you to create manual and automatic average faces and also to recognise them in a database. To do that, you just need to select what you want to do in the main menu and start working! The main menu is divided in three independent parts: two for face creation and a last one to detect a face as you can see in the next figure.

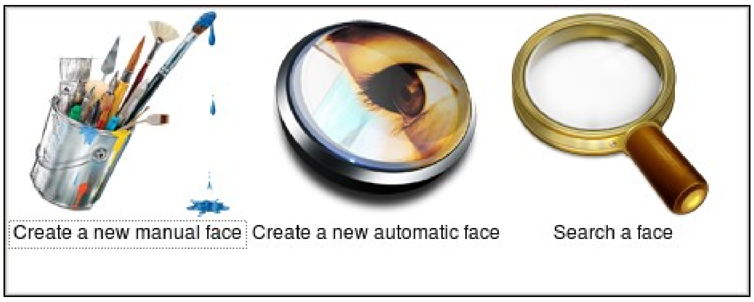


FIGURE D.1: Screenshot. Welcome menu.

D.1.1 Average face creation

1. **Manual** This method will offer you the possibility of designing node by node the positions of the triangular mesh edges. You will be able to modify one by one the images and decide which are the most important features you want to highlight (such as the eyes, nose or mouth). The main disadvantage of this method is the time expended. You will need approximately (depends on the level of detail and the patience of the person who is producing the images) about two minutes to create a warped image.

2. **Automatic** This alternative method offers the possibility of generating automatically an average image introducing a large batch of images. The advantage is the speed: more than 20 images in less than a second. The problem is that because it is an automated procedure, only those pictures where both eyes, mouth and nose are detected will be accepted. Otherwise, the face image will be rejected. That means that an important number of faces would be rejected due to light conditions or extreme non-frontal position of the face.

D.1.2 Database search

The last of the three options of the main menu. This form will allow you to recognise a face using eigenfaces and a face database created previously.

D.2 Manual face creation

This first part of the application has been designed to involve in the process the user as much as possible. For this reason, all the menus and options have been decided to be clear and accurate. As soon as we select this option in the main menu, a complete new environment will be displayed.

This form is defined by 5 parts:

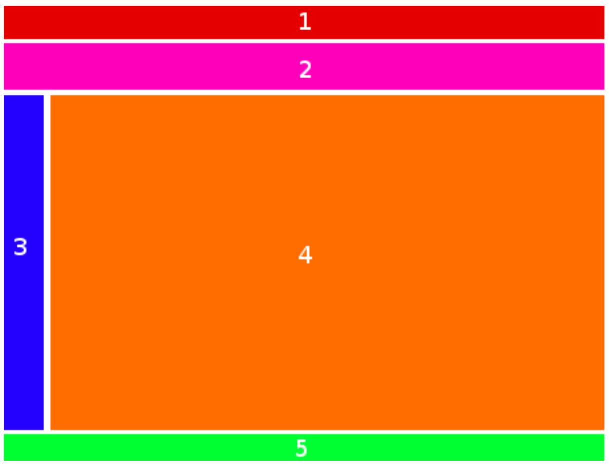


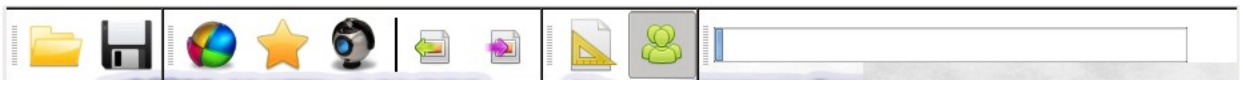
FIGURE D.2: Layout of the manual face creator

D.2.1 Text menu

Contains all the options that the software offers. It is divided in 6 parts.Each one focused on a different area.

- **File** Open a batch of images, save the actual one, close the actual image and also close the application.
- **View** Show the mesh lines inside the picture and display the help face on the right area of 4
- **Mesh options** Move the nodes individually, in a group etc...
- **Operations** When the user wants to create an average or see how the actual mesh has been deformed yet.
- **Images** Control of the batch of images loaded. Move to the next face or return to the previous one.
- **Help** Information about the author and the libraries used.

D.2.2 Main options toolbar



This toolbar contains the top used commands in order to save time navigating by the menus of the previous section. The options are displayed in Table D.2.2

D.2.3 Secondary options toolbar

The purpose of this secondary toolbar is to play with the nodes when an image is loaded. To facilitate the management of the nodes, this toolbar has been divided in three different modules:

D.2.3.1 Single node administration

With the buttons provided, the user can move the selected node up, down, right and left 5 pixels each time. Pressing directly the keyboard keys "up" "down" "left" or "right" the same effect will be done.

Icon	Description
📂	**Open** - Open a batch of images to be treated individually by the user.
💾	**Save** - Save the actual image with the actual mesh.
●	**Warp** - In case the user wants to obtain an image of the deformation created before saving the image this option will generate it and shows it in an independent window.
⭐	**Average** - If the user wants to create an average of a batch of warped images this option will ask for an input group and a place to save the average image generated.
📷	**Webcam** - Do you need to acquire an image in real time? This option lets the user to capture an image from a connected webcam and treat it in the main window directly.
⬅	**Previous image** - Deformation of the actual image with the mesh designed and turn back to the previous face in the list.
➡	**Next image** - Deformation of the actual image with the mesh designed and advance to the next face in the list.
📐	**Show edges** - Display or not the mesh lines in order to provide the best distribution of the nodes. This option can help sometimes to understand the deformation of some areas of the image.
👥	**Help view** - Display or hide the right face that highlights the node selected. This option is quite useful to understand which node is being selected.

Table D.1: Description of the toolbar icons in the Manual Form

D.2.3.2 Group of nodes administration

With the first four buttons, the user can move the selected group of nodes up, down, right and left 5 pixels each time. With the next two buttons, the group will turn clock or anticlockwise. With the last two buttons, the distance between nodes in the group can be increased or decreased.
Groups are created by colour and they are provided in the table 1. Pressing directly the keyboard keys "a" for left movement, "d" for right movement, "w" for up movement or "s" for down movement the same effect would be done

Colour	Description
Blue	Outline (jaw, ears, pinna, temples, scalps and crown)
White	Chin
Orange	Left eyebrow
Turquoise	Right eyebrow
Pink	Right eye
Yellow	Left eye
Red	Mouth
Green	Nose
Violet	Other landmarks like cheeks or forehead

Table D.2: Landmarks/nodes referenced by colour in groups

D.2.3.3 All nodes administration

With these four buttons the user can move all the nodes at the same time. This option is very useful when the face is displaced inside the image a certain distance.

Pressing directly the buttons ”f” for left movement, ”h” for right movement, ”t” to move all nodes up and ”g” to move them down.

D.2.4 Working Area

This zone corresponds to the area where the user interacts with the application during most of the time. Here the face image is plotted on the left and (optionally) a help face is represented on the right to understand the nodes selected. Because this area is a graphic area, some extra features can be done.

1. If the user launched the software from a terminal, the nodes position can be displayed in the terminal pressing key ”m”.
2. The key ”l” displays / hide the actual mesh
3. Scrolling the mouse over a picture, a zoom in/out will be done

All nodes can be moved using the toolbar, keyboard or **dragging and dropping them!**

D.2.4.1 Status bar

This bar indicates the tips and descriptions of each one of the objects inside the window and other messages as warnings or progress.

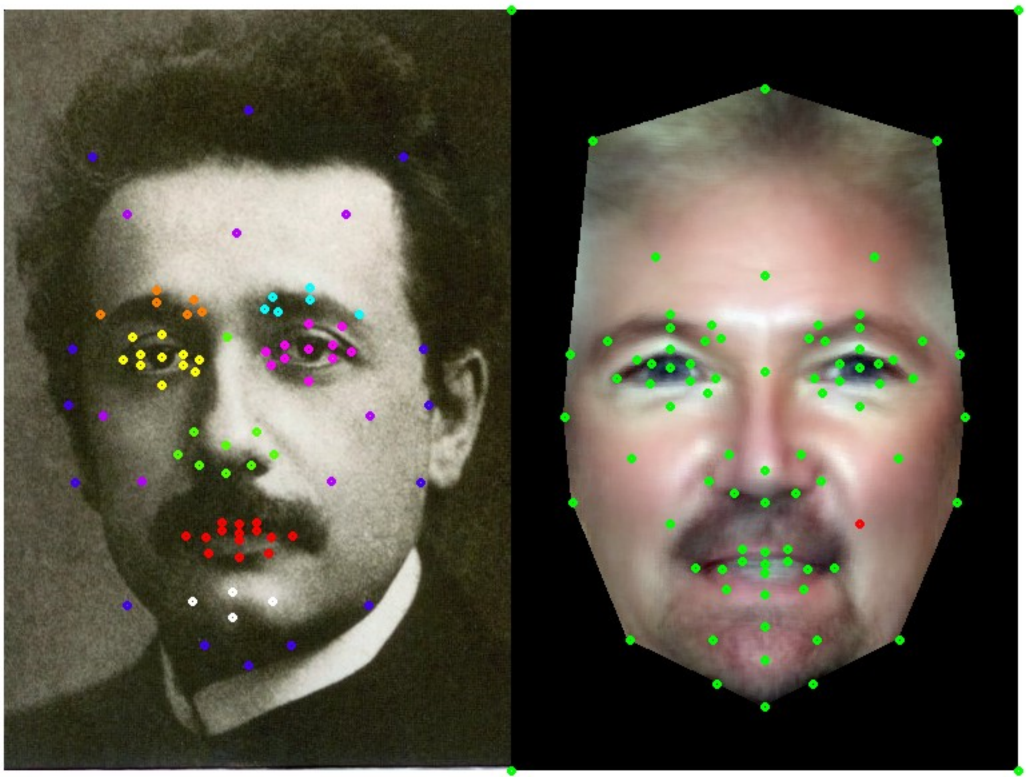
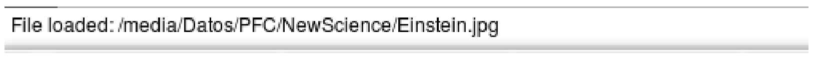


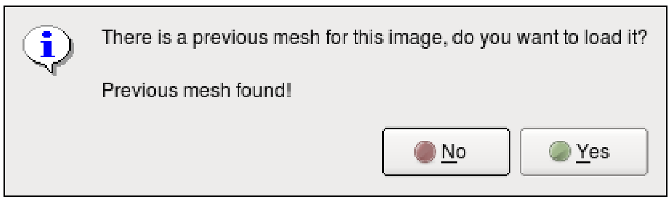
FIGURE D.3: Example of a real face processing. The left area contains the desired warping image while the right one contains the help face with a red node advising us the last node selected



File loaded: /media/Datos/PFC/NewScience/Einstein.jpg

These messages will appear just during a period of two or three seconds to alert the user about a new situation. If an important error happen or an question is generated, the software will create a message dialog to interact with the user. This messages are described in the next point.

During the life of the application several messages can appear on the screen. Each one



There is a previous mesh for this image, do you want to load it?

Previous mesh found!

of them has a different purpose. Most relevant are enumerated in the following list with a short explanation.

Message	Why?
Original mesh file %1 is missing. Cannot read the file.	When doing an average of a group of warped images, it's necessary to have in the same path as each image a file with the coordinates of the nodes of the warped mesh. For each face, the name will be **originalname.jpg-warped.dat** in case there is a previous mesh and **pos.dat** by default.
Help mesh file %1 is missing! Cannot read the file.	It's necessary to have in the same path as the application a file with coordinates of the nodes for the help face on the right. The name is **posHelpNodes.dat**
Frontal mesh file %1 is missing! Cannot read the file.	It's necessary to have in the same path as the application the file with coordinates of the nodes for the frontal mesh. The name is **pos.dat**
Triangles file %1 is missing! Cannot read the file.	It's necessary to have in the same path as the application the file with coordinates of the nodes for the frontal mesh. The name is **pos.dat**
Error saving mesh file: %1 Do you have rights to do that?	It is strange, but sometimes an error saving the image could be produced. Normally this error is due to lack of space in the hard disk, network problems or lack of privileges to write in a certain folder
Fatal error warping the mesh. It seems that there are some points of the mesh that cannot be reached. Exiting...	When processing a warping process is possible to find some nodes out of the image range. Normally that situation happens when the user drop the nodes out of the image region. This problem causes a fatal error and exits the program.
Are you sure you want to leave the actual work and exit?	When the user tries to close the application a message will appear to ensure that this is the desired selection. All the work that wasn't saved will be lost.
Are you sure you want to remove actual work without saving?	When the user click on Close menu, an alert will ask if this is the desired option.
Congratulations! You have reached the end of the list of pictures!	When the user finishes the whole batch of images, the application understands that the user wants to create an average. That is why this option is offered afterwards.
Cannot read file %1:%2.	If the user tries to open a corrupt image, this alert will advise the causes and the file involved in the problem.
There is a previous mesh for this image, do you want to load it? Mesh found!	When the user selects an image that was already warped, the application offers to resume the state of the nodes, reallocating them in the position readed from the file. This file will be **originalname.jpg-warped.dat**

Table D.3: List of most frequent messages displayed by the application

D.3 Automatic face creation

The second module of the application consist on a simple form where first the user is asked for a batch of images as an input. Then some information is displayed about the images selected and afterwards some output parameters can be selected.

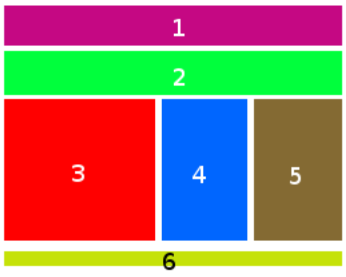
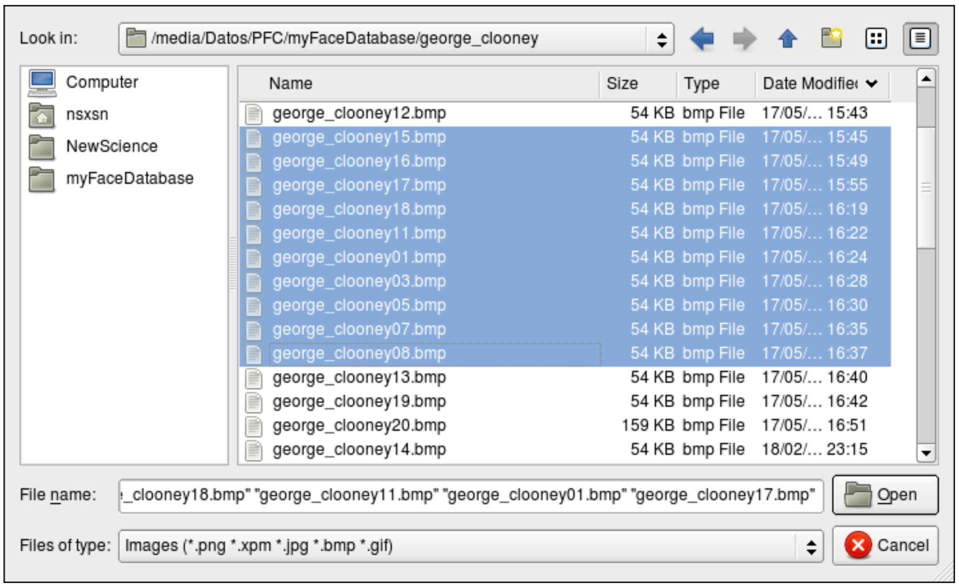


FIGURE D.4: Layout of the automatic face creator

D.3.1 Face batch selection

This is the first step. When the button Search for face files.. is pressed an external window will appear waiting for a selection of the images to load. In the next figure the dialog is displayed. It is important to notice that is possible to select more than one image at the same time. That means, it is multi- selection.



Look in: /media/Datos/PFC/myFaceDatabase/george_clooney

Name	Size	Type	Date Modified
george_clooney12.bmp	54 KB	bmp File	17/05/... 15:43
george_clooney15.bmp	54 KB	bmp File	17/05/... 15:45
george_clooney16.bmp	54 KB	bmp File	17/05/... 15:49
george_clooney17.bmp	54 KB	bmp File	17/05/... 15:55
george_clooney18.bmp	54 KB	bmp File	17/05/... 16:19
george_clooney11.bmp	54 KB	bmp File	17/05/... 16:22
george_clooney01.bmp	54 KB	bmp File	17/05/... 16:24
george_clooney03.bmp	54 KB	bmp File	17/05/... 16:28
george_clooney05.bmp	54 KB	bmp File	17/05/... 16:30
george_clooney07.bmp	54 KB	bmp File	17/05/... 16:35
george_clooney08.bmp	54 KB	bmp File	17/05/... 16:37
george_clooney13.bmp	54 KB	bmp File	17/05/... 16:40
george_clooney19.bmp	54 KB	bmp File	17/05/... 16:42
george_clooney20.bmp	159 KB	bmp File	17/05/... 16:51
george_clooney14.bmp	54 KB	bmp File	18/02/... 23:15

File name: _clooney18.bmp" "george_clooney11.bmp" "george_clooney01.bmp" "george_clooney17.bmp"

Files of type: Images (*.png *.xpm *.jpg *.bmp *.gif)

FIGURE D.5: Multi-selection of face images

D.3.2 Output configuration

The program offers as an option to modify the standard conditions of the output image generated.

- **Size** By default the images will be generated at 180x180 if the text boxes are blank, otherwise the introduced values will be taken into account.

- **Grey scale** It is possible to convert the output image into a grey scale image.

$\boxed{\text{The grey scale algorithm used is just as the average of R + G + B for each pixel.}}$

- **Save path** A button that launches a dialog window to select the name of the image that is going to be created.

D.3.3 Current information

During the scanning of images and the automated process, some information of each face is displayed such as the position of the image in the list, the number of colours and the size of the original image. In the figure 6 an example of real time process is demonstrated.

D.3.4 Input intermediate image before warping

This image contains the original image before it is warped. Just resizing is done.

D.3.5 Output intermediate image after warping

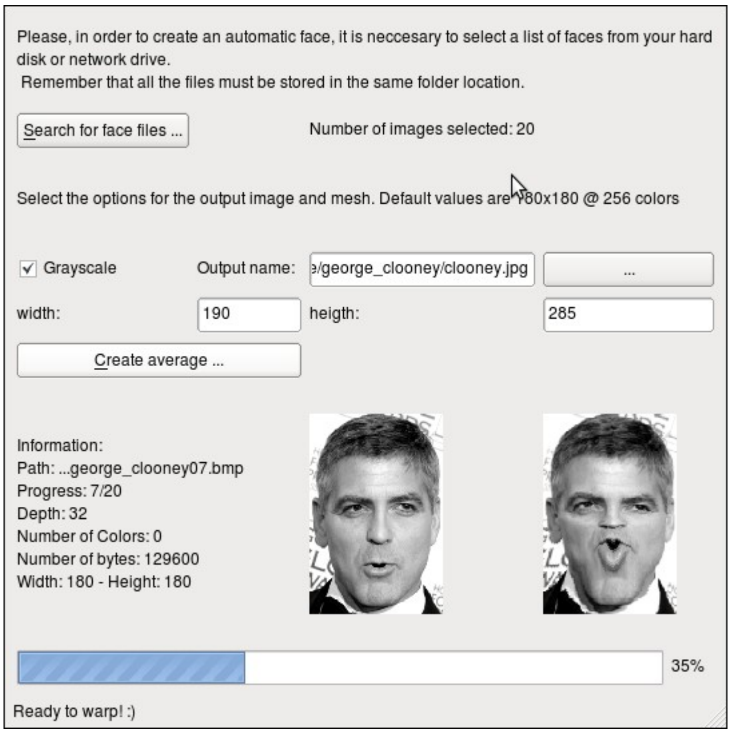
This image contains the final individual image after it is warped automatically. It is easy to compare them and find the differences for each face.

D.3.6 Progress

The progress bar represents the number of faces processed over the total number of faces in the list. This object can display easyly the real state of the process and internal subroutines.

D.4 Face recognition. Search a face.

The last module of the application is designed to complete the investigation. Here the user can provide a face database and the software will find by itself the closest face inside the database introducing a given face. The database must be introduced following a valid eigenfaces list format (xml file with standard tags).


Please, in order to create an automatic face, it is neccesary to select a list of faces from your hard disk or network drive.
Remember that all the files must be stored in the same folder location.

Number of images selected: 20

Select the options for the output image and mesh. Default values are 180x180 @ 256 colors

☑ Grayscale

Output name:	e/george_clooney/clooney.jpg		
width:	190	heigth:	285

Information:
Path: ...george_clooney07.bmp
Progress: 7/20
Depth: 32
Number of Colors: 0
Number of bytes: 129600
Width: 180 - Height: 180

35%

Ready to warp! :)

FIGURE D.6: Automatic face averaging

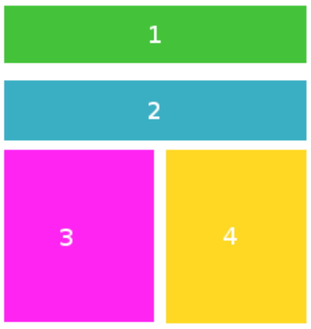


FIGURE D.7: Layout of the search face form

D.4.1 Database selection

Introduce the path of the database file. Some information about the eigenfaces stored inside the database is presented, also the number of images and the date of creation of the database.

D.4.2 Face to recognise selection

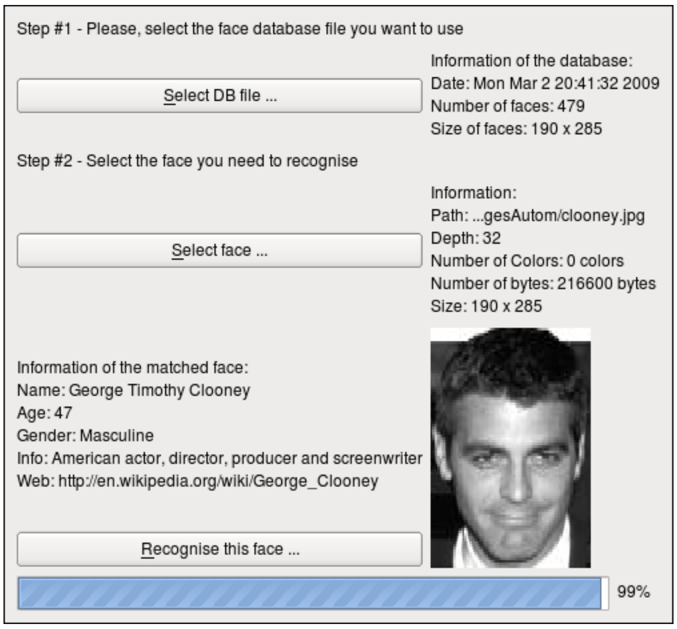
Introduce the path of the face to recognise after pressing the button Select face. Some information about the image is presented (number of colours, size, number of bytes, etc...)

D.4.3 Matched face report

Once the image is selected and the corresponding button is pressed (Recognise this face) and the application has found the closest face, some information about the individual will be displayed (age, profession, real name or link to find more information)[1].

D.4.4 Matched face

This image contains the resulting image. This image will match with the closest person of the database.



Step #1 - Please, select the face database file you want to use

Information of the database:
Date: Mon Mar 2 20:41:32 2009
Number of faces: 479
Size of faces: 190 x 285

Step #2 - Select the face you need to recognise

Information:
Path: ...gesAutom/clooney.jpg
Depth: 32
Number of Colors: 0 colors
Number of bytes: 216600 bytes
Size: 190 x 285

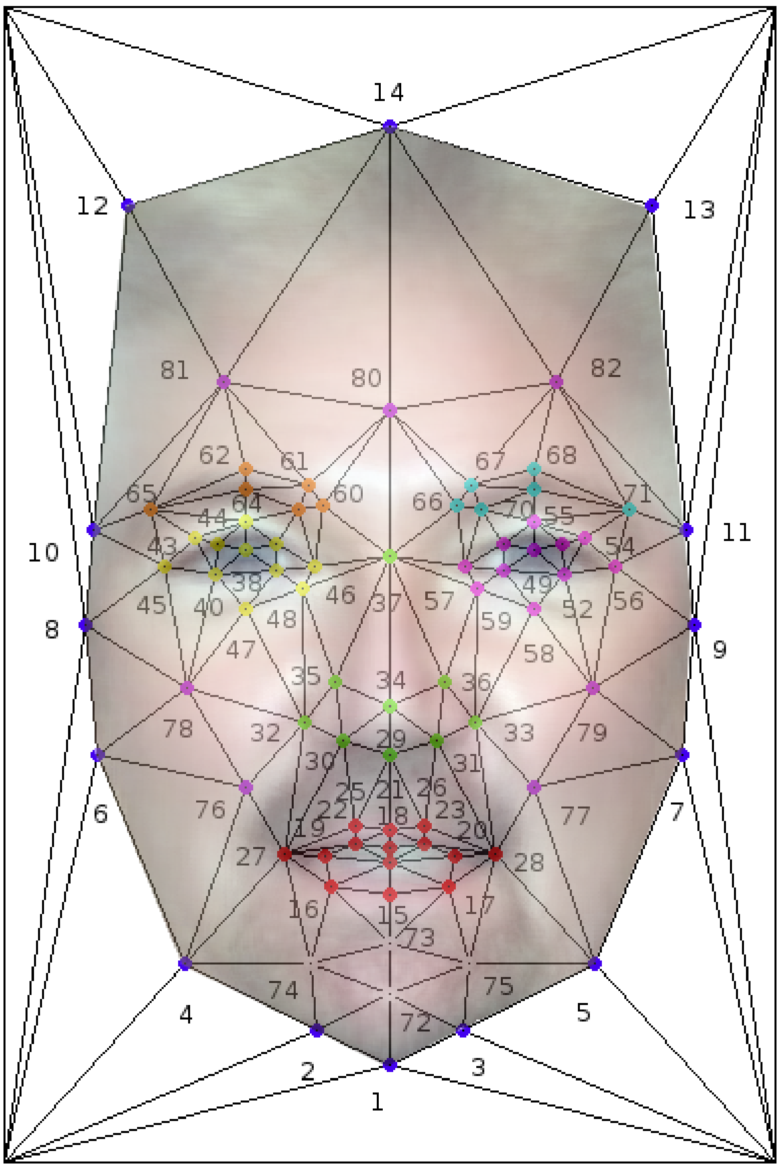
Information of the matched face:
Name: George Timothy Clooney
Age: 47
Gender: Masculine
Info: American actor, director, producer and screenwriter
Web: http://en.wikipedia.org/wiki/George_Clooney

FIGURE D.8: Example of face recognition in real time

[1]All this information has been extracted from `www.wikipedia.org`

Appendix E

Manual face mesh

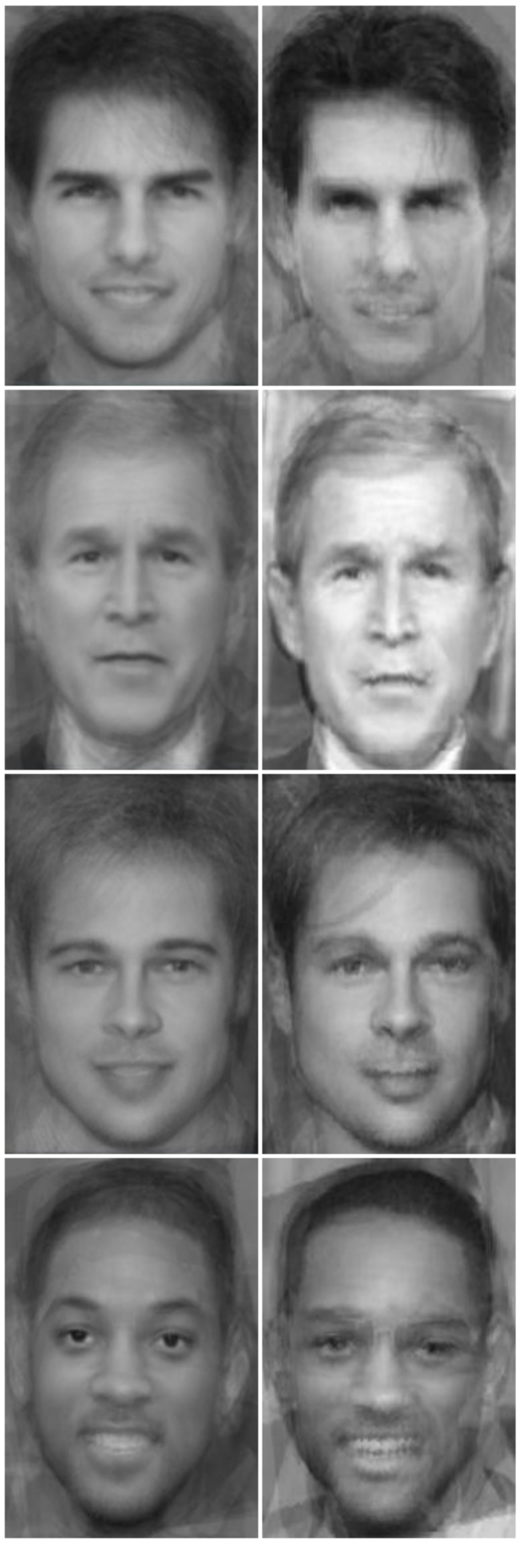


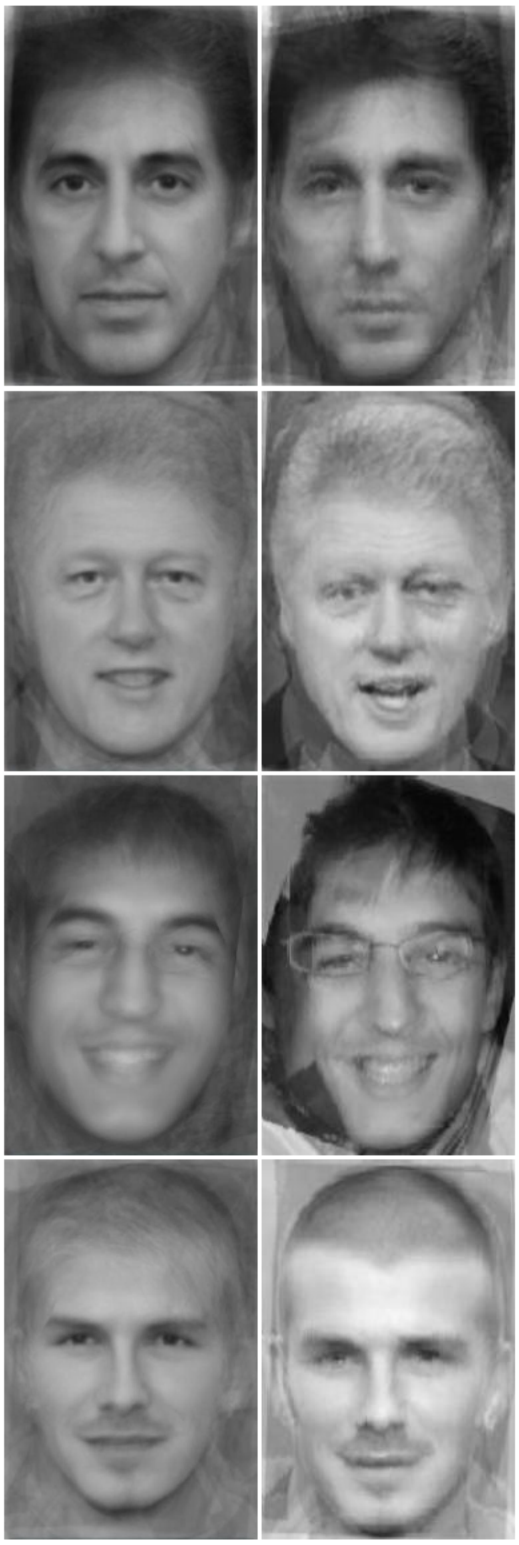
REGION1	Outline		
Number	Description	X	Y
1	Chin centre	186	522
2	Chin left	148	505
3	Chin right	219	506
4	Jaw left	85	472
5	Jaw right	287	472
6	Ear left	42	369
7	Ear right	330	366
8	Pinna left	37	301
9	Pinna right	337	310
10	Temple left	41	258
11	Temple right	333	257
12	Scalp left	51	100
13	Scalp right	309	97
14	Crown	190	59
REGION2	Mouth		
Number	Description	X	Y
15	Blip$_BC$	185	438
16	Blip$_BL$	156	434
17	Blip$_BR$	214	435
18	Blip$_TC$	186	422
19	Blip$_TL$	152	419
20	Blip$_TR$	215	419
21	Tlip$_BC$	187	415
22	Tlip$_BL$	169	413
23	Tlip$_BR$	203	413
24	Tlip$_TC$	187	406
25	Tlip$_TL$	169	404
26	Tlip$_TR$	203	404
27	Lips left	136	417
28	Lips right	241	419
REGION3	Nose		
Number	Description	X	Y
29	Septum	187	369
30	Left nostril	164	360
31	Right nostril	210	363
32	Left flange	145	353
33	Right flange	228	353
34	Tip	186	345
35	Left valley	160	334
36	Rightvalley	214	332
37	Bridge	188	271

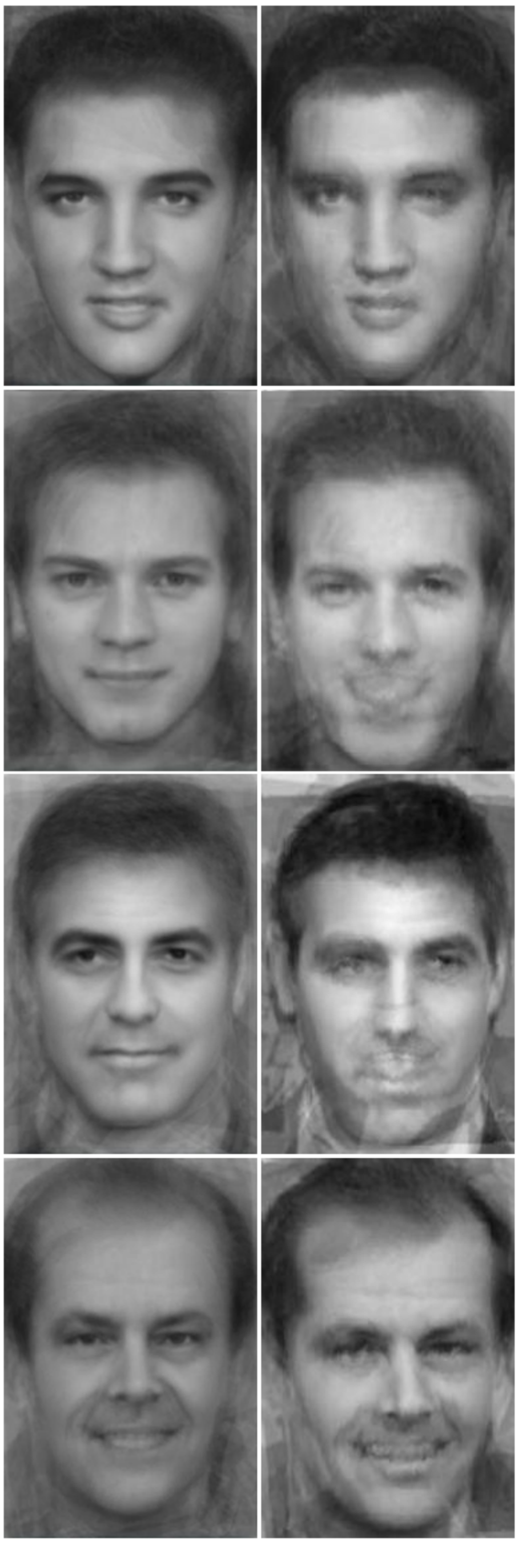
REGION4	Eyes		
Number	Description	X	Y
38	Lpupil	118	268
39	Liris$_T O$	105	266
40	Liris$_T I$	133	265
41	Llid$_L O$	105	280
42	Llid$_L I$	133	279
43	Llid$_T O$	94	261
44	Llip$_T I$	118	254
45	Louter fold	78	275
46	Ltear duct	153	276
47	Lbag outer	118	296
48	Lbag inner	148	285
49	Rpupil	260	268
50	Riris$_T O$	276	265
51	Riris$_T I$	245	265
52	Rlid$_L O$	276	280
53	Rlid$_L I$	245	278
54	Rlid$_T O$	286	262
55	Rlip$_T I$	260	254
56	Router fold	300	276
57	Rtear duct	227	278
58	Rbag outer	261	297
59	Rbag inner	234	289
REGION5	Eyebrows		
Number	Description	X	Y
60	Lbrow inner	157	247
61	Lbrow TI	148	237
62	Lbrow TM	118	229
63	Lbrow LI	145	249
64	Lbrow LM	118	239
65	Lbrow outer	74	249
66	Rbrow inner	222	246
67	Rbrow TI	229	235
68	Rbrow TM	261	228
69	Rbrow LI	235	247
70	Rbrow LM	260	238
71	Rbrow outer	309	248
REGION6	Landmarks		
Number	Description	X	Y
72	Chin tip	186	487
73	Chin top	186	462
74	Chin left	147	472
75	Chin right	224	472
76	Line left	113	385
77	Line right	255	385
78	Cheek left	84	334
79	Cheek right	285	337
80	Forehead centre	190	199
81	Forehead left	105	185
82	Forehead right	270	185

Appendix F

Complete list of average faces







Bibliography

[1] Face Recognition Homepage. General website especialized in face recognition news. Created by Mislav Grgic and Kresimir Delac. `http://www.face-rec.org/general-info` last visited April. 10, 2009.

[2] D., Panica, S., Banciu, D.; Negru, V.; Eckstein, A. Optical Character Recognition on a Grid Infrastructure. *Petcu*: Tables 1,2 ,p. 23, 2008.

[3] C. J. Hawthorn, K. P. Weber, and R. E. Scholten. A novel method of recognizing ageing face based on EHMM. *Proceedings of the Fourth International Conference on Machine Learning and Cybernetics*, Guangzhou. p. 4599–4609, August 2005.

[4] W. Zhao, R. Chellappa, P. Phillips, and A. Rosenfeld. Face recognition: A literature survey. *ACM Computing Surveys*, p 399–458, 2003.

[5] W. W. Bledsoe. The model method in facial recognition. *Panoramic Research Inc.*, Palo Alto, CA. August, 1966.

[6] Rana Islam (National Policing Improvement Agency). Facial Image National Database (FIND) *Police Standard for Still Digital Image Capture and Data Interchange of Facial/Mugshot and Scar, Mark Tattoo Images.* p. 9–28 May, 2007. `http://www.npia.police.uk/en/docs/Capture_interchange_standard_Facial_SMT_images.pdf` last visited April. 10, 2009.

[7] Robert O'Harrow. Matching Faces with Mugshots: Software for Police, Others Stir Privacy Concerns. *Washington Post*, July 31, 2001.

[8] P. J. Phillips, H. Moon, P. J. Rauss, and S. Rizvi. The FERET Evaluation Methodology for Face-Recognition Algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 22, No. 10. p. 1090–1104, 2000.

[9] Enterprise Access website. Identity Solutions Inc. 2006-2009. `http://www.bioscrypt.com`.

[10] R. C. Gonzalez and R. E. Woods. Digital Image Processing 2nd Edition. Prentice Hall, New Jersey, $2^n d$ edition, 2002.

[11] Konstantinos N. Plataniotis, Anastasios N. Venetsanopoulos, A. Lacroix. Color image processing and applications. Springer, p. 259, 2000.

[12] Albayrak, Songul. Color Quantization by Modified K-Means Algorithm. *Journal of Applied Science*, Vol. 1, Issue 4, p. 508–511, 00/2001.

[13] Castleman, Kenneth R. *Digital Image Processing* 1st ed., Chap.21 p 550–554, 1996.

[14] Tang Yang; Wu Hui-zhong; Xiao Fu; Xiao Liang. Inverse image warping without searching. *Control, Automation, Robotics and Vision Conference*, Issue , p. 386–390 Vol. 1, 6-9 Dec. 2004.

[15] George Wolberg. Recent Advances in Image Morphing. *Proc. Computer Graphics Intl.*,p. 64–71. Pohang, Korea, June 1996.

[16] Thaddeus Beier, Shawn Neely. Feature-Based Image Metamorphosis. *Proc. of the 19th Annual Conference on Computer Graphics and Interactive Techniques*, Vol. 26, p. 35–42, 1992.

[17] Prashant K. Oswal and Prashanth Y. Govindaraju. Morphing: A Comparative Study *Internal publications* Department of Electrical and Computer Engineering, Clemson University, Clemson. p. 1–5, 2004.

[18] Hui Li, Hui Lin, Guang Yang. A New Facial Expression Analysis System Based on Warp Image *Proceedings of the 6th World Congress on Intelligent Control and Automation (WCICA)* Vol.2, p. 10045–10049, June 2006

[19] Ngo Quoc Tao, Nguyen Due Long. A novel approach for image morphing. *IEEE Asia-Pacific Conference on Circuits and Systems* p. 97–100, December 2004.

[20] Su Zang, Hanfeng Chen, Pengfei Shi Mosaic and Warping for Forward Moving Images *Proceedings of the Computer Graphics International 2001* p. 363–366, July 2001.

[21] Castleman, Kenneth R. *Digital Image Processing* 1st ed., Chap.8 p. 127, 1996.

[22] Pratt, William K. PixelSoft Inc. Digital Image Processing.4th ed. 2007 *Fundaments of Image Processing*,p. 419–664.

[23] Viola, P., Jones, M. Rapid Object Detection using a Boosted Cascade of Simple Features *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference* Vol. 1, p. 511–518, 2001.

[24] A. Jacquin and A. Eleftheriadis. Automatic location tracking of faces and facial features in video signal. *Proceedings of International Workshop on Automatic Face and Gesture Recognition* p. 142–147, 1995.

[25] D. Chai and K. Ngan. Locating facial region of head and shoulders color image. *Proceedings of Automatic Face and Gesture Recognition* p. 124–129, 1998.

[26] Matthew A. Turk, Alex P. Pentland Face Recognition Using Eigenfaces *IEEE Conf. on Computer Vision and Pattern Recognition*, p. 586–591, 1991.

[27] Matthew A. Turk, Alex P. Pentland EigenFaces for recognition *Journal of Cognitive Neuroscience* Vol 3, number 1. p. 71–84, 1991.

[28] Thomas Heseltine, Nick Pears, Jim Austin. Evaluation of image pre-processing techniques for eigenface based face recognition *The Proceedings of the Second International Conference on Image and Graphics, SPIE* Vol. 4875, p. 677–685, 2002.

[29] J. Lu, K.N. Plataniotis, and A.N. Venetsanopoulos. Regularized Discriminant Analysis For the Small Sample Size Problem in Face Recognition *Pattern Recognition Letters* Vol. 24, Issue 16, p. 3079–3087, 2003.

[30] Laurenz Wiskott, Jean-Marc Fellous, Norbert Krger, Christoph von der Malsburg. Face Recognition by Elastic Bunch Graph Matching *Intelligent Biometric Techniques in Fingerprint and Face Recognition* Chapter 11, p. 355–396, 1999.

[31] Brad Calder, Dirk Grunwald, Benjamin Zorn. Quantifying behavioral between C and C++ programs. *Journal of Programming Languages* Vol.2, p. 313–351, 1996.

[32] Matthias Kalle Dalheimer. Programming with Qt. *O'Reilly UK*, 2002.

[33] Ward M. Computer Vision notes. Worcester Polytechnic Institute.
`http://davis.wpi.edu/~matt/courses/morph/2d.htm`

[34] M. Castrillón-Santana, O. Déniz-Suárez, L. Antón-Canalís and J. Lorenzo-Navarro Face and Facial Feature Detection Evaluation. Performance Evaluation of Public Domain Haar Detectors for Face and Facial Feature Detection. *The International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications* January 2008 `http://www.visapp.org/` last visited April. 10, 2009

[35] Gary Bradski, Adrian Kaehler. Learning OpenCV. Computer Vision with the OpenCV Library. O'Reilly, 1^st Ed. 2008, p. 506–515

[36] Scott Hassan, Steve Cousins, Eric Berger, Keenan Wyrobek Willow Garage. Robotic Research Institute. `http://opencv.willowgarage.com` last visited April. 13, 2009

[37] R. Jenkins, A.M. Burton. 100% Accuracy in Automatic Face Recognition. *Science Magazine* Vol. 319, p. 435, January 2008

[38] R. Jenkins, A.M. Burton, D. White. Face Recognition from Uncostrained Images: Progress with Prototypes. *Automatic Face and Gesture Recognition (FGR 2006) 7th International* p. 25–30, April 2006

[39] A.M. Burton,R. Jenkins, Peter J.B Hancock, D. White. Robust representations for face recognition: The power of averages. *Congnitive Psychology* Vol.51 Issue 3, p. 256–284, November 2005

[40] Walter Mora F., Alex Borbón A. Edición de textos cientícos en LaTeX. *Revista Digital Matemática Educación e Internet* `http://www.cisde.itcr.ac.cr`

Source Code - StoneFace v.1.0

```
// Master Thesis - StoneFace v.1.0 beta C by Jorge Garcia Bueno
// Universidad Carlos III de Madrid + University of Glasgow 2009
// All rights reserved by the author
// -----------------------------------------------------------------
// automaticwidget.h - Definitions of automaticwidget.cpp

#ifndef AUTOMATICWIDGET_H
#define AUTOMATICWIDGET_H

#include <QtGui>
#include <QPixmap>
#include <QGraphicsView>
#include <QListView>
#include <QDialog>
#include <QPushButton>
#include <QDesktopWidget>
#include "mainwindow.h"
#include "stonefacewidget.h"
#include "mixeraut.h"


class QAction;
class QMenu;
class QGraphWidget;
class WelcomeWidget;
class MixerAutomatic;
class StoneFaceWidget;

class AutomaticWidget : public QWidget
{
    Q_OBJECT


private:
    // MainWindow handle
    QMainWindow *mainWin;
    QProgressBar *pb;
    QPixmap *actualPic;
    QPixmap *warpedPic;
    QLabel *instructionsLbl;
    QLabel *instructionsLbl2;
    QLabel *numFaces;

    QLabel *mixInfo;
    QCheckBox *cb;
    QPushButton *buttonStartMix;
    QPushButton *buttonStartSearch;
    QPushButton *buttonLookPath;
    QLabel *face1;
    QLabel *face2;
    StoneFaceWidget *widget;
    MixerAutomatic *mixer;

    QLineEdit *imgname;
    QLineEdit *imgw;
    QLineEdit *imgh;

    void StartInterface();
    void MoveCenterScreen();

    QGridLayout *lLayout;

private slots:
    void CreateMix();
    void SearchFiles();
    void LookPath();
public:

    AutomaticWidget(QMainWindow *parent);
    void ShowStatus();
```

```
public slots:

    // Status
    void ProgressBarSetValue(int value);
    void SetStatus(QString,int,int,int,int,int,int,int, QImage, QImage);
    void zoneScanned(QString);
};

#endif


// Master Thesis - StoneFace v.1.0 beta C by Jorge Garcia Bueno
// Universidad Carlos III de Madrid + University of Glasgow 2009
// All rights reserved by the author
// ---------------------------------------------------------------
// automaticwidget.cpp - Interface to control the creation of automatic pictures

#include "automaticwidget.h"

AutomaticWidget::AutomaticWidget(QMainWindow *parent)
{
    // Remember the handle of the parent for the returning function
    mainWin = parent;

    // Start the interactive objects
    StartInterface();
    MoveCenterScreen();
    mixer = NULL;
    widget = NULL;
}

void AutomaticWidget::CreateMix()
{
    if (mixer != NULL && widget != NULL)
    mixer->Start(imgname->text(),imgw->text().toInt(),imgh->text().toInt(),cb->isChecked() );
    delete widget;
    delete mixer;
}
void AutomaticWidget::MoveCenterScreen()
{

    int screenWidth, width;
    int screenHeight, height;


    QDesktopWidget *desktop = new QDesktopWidget();
    QSize windowSize;
    screenWidth = desktop->width(); // get width of screen
    screenHeight = desktop->height(); // get height of screen

    windowSize = size(); // size of our application window
    width = windowSize.width();
    height = windowSize.height();

    move((screenWidth-width)/2,(screenHeight - height)/2);
}
void AutomaticWidget::StartInterface()
{
    instructionsLbl = new QLabel("Please, in order to create an automatic face, it is neccesary to select
a list of faces from your hard disk or network drive.\n Remember that all the files must be stored in the
same folder location.", this);
    instructionsLbl->setWordWrap(true);
    instructionsLbl2 = new QLabel("Select the options for the output image and mesh. Default values are 18
0x180 @ 256 colors", this);
    instructionsLbl2->setWordWrap(true);
    numFaces = new QLabel("Number of images selected: 0");
    QString mixInfoL("");
    mixInfoL += "Information:\nPath: ...\n";
    mixInfoL += "Progress: \n";
    mixInfoL += "Depth: \n";
    mixInfoL += "Number of Colors: \n";
    mixInfoL += "Number of bytes: \n";
    mixInfoL += "Width:    - Height: ";
    mixInfo = new QLabel(mixInfoL);

    buttonStartSearch = new QPushButton("&Search for face files ...", this);
    connect(buttonStartSearch, SIGNAL(clicked()), this, SLOT( SearchFiles() ) );

    buttonStartMix = new QPushButton("&Create automatic average ...", this);
    connect(buttonStartMix, SIGNAL(clicked()), this, SLOT(CreateMix()));

    buttonLookPath = new QPushButton("...", this);
    connect(buttonLookPath, SIGNAL(clicked()), this, SLOT(LookPath()));
    face1 = new QLabel("Virtual Face",this);
    face1->setPixmap(QPixmap("./images/user.png"));

    face2 = new QLabel("Virtual Face",this);
    face2->setPixmap(QPixmap("./images/user.png"));

    // Creation of layout to define the positions of the objects

    pb = new QProgressBar(this);
    cb = new QCheckBox("Grey Scale",this);
    imgname = new QLineEdit(this);
    imgw = new QLineEdit(this);
    imgh = new QLineEdit(this);
    imgname->setText("average.jpg");
    imgw->setText("190");
    imgh->setText("285");
    lLayout = new QGridLayout;
    lLayout->addWidget(instructionsLbl,0,0,1,4);
    lLayout->addWidget(buttonStartSearch,1,0,1,2);
    lLayout->addWidget(numFaces,1,2,1,1);

    // ------------------------------------------

    lLayout->addWidget(instructionsLbl2,2,0,1,4);
    lLayout->addWidget(cb, 3 ,0,1,1);

    lLayout->addWidget(new QLabel("Output name:",this), 3,1,1,1);
    lLayout->addWidget(imgname,3,2,1,1);
    lLayout->addWidget(buttonLookPath, 3,3,1,1);
    lLayout->addWidget(new QLabel("Width:",this),4,0,1,1);
    lLayout->addWidget(imgw,4,1,1,1);
    lLayout->addWidget(new QLabel("Height:",this),4,2,1,1);
    lLayout->addWidget(imgh,4,3,1,1);

    lLayout->addWidget(buttonStartMix,5,0,1,2);

    // ------------------------------------------

    lLayout->addWidget(mixInfo, 6 ,0,3,2);


    lLayout->addWidget(face1,6,2,3,1);
    lLayout->addWidget(face2,6,3,3,1);

    lLayout->addWidget(pb, 9, 0,1,4);

    setLayout(lLayout);

}
```

```
void AutomaticWidget::SearchFiles()
{
    int c;

    // Create the original components
    widget = new StoneFaceWidget();
    mixer =  new MixerAutomatic(widget, this);

    connect(mixer, SIGNAL(SetStatus(QString,int,int,int,int,int,int,int, QImage, QImage)),
            this,    SLOT(SetStatus(QString,int,int,int,int,int,int,int, QImage, QImage)));
    connect(widget, SIGNAL(zoneScanned(QString)), this, SLOT(zoneScanned(QString)) );

    c = mixer->SearchFiles();
    numFaces->setText(QString("Number of images selected: ") + QString::number(c));
}


// External methods to update the status

void AutomaticWidget::ProgressBarSetValue(int value)
{
    //pb->setValue(value);
}

void AutomaticWidget::SetStatus(QString a,int order,int total,int depth,int numColors,int numBytes,int wid
th,
                                int height, QImage original, QImage morphed)
{

QString info("");
QPixmap qpx;
//repaint();
info += "Information:\nPath: ...";
info +=  a.right(20) + "\n";
info += "Progress: " + QString::number(order) + "/" + QString::number(total) + "\n";
info += "Depth: " + QString::number(depth) + "\n";
info += "Number of Colors: " + QString::number(numColors) + "\n";
info += "Number of bytes: " + QString::number(numBytes) + "\n";
info += "Width: " + QString::number(width) + " - Height: " + QString::number(height);

mixInfo->setText(info);
face1->setPixmap(qpx.fromImage(original.scaled(100,150)));
face2->setPixmap(qpx.fromImage(morphed.scaled(100,150)));
pb->setValue(100*order/total);

update();

}


void AutomaticWidget::zoneScanned(QString part)
{
    mainWin->statusBar()->showMessage(QString("Scanning ... ") + part);

}

void AutomaticWidget::LookPath()
{
    QString fileName = QFileDialog::getSaveFileName(this, tr("Select path to save the average file"),
                                                    "./", tr("Images (*.png *.xpm *.jpg *.bmp)"));

    if (fileName.length() != 0)
        imgname->setText(fileName);

}


// Master Thesis - StoneFace v.1.0 beta C by Jorge Garcia Bueno
// Universidad Carlos III de Madrid + University of Glasgow 2009
// All rights reserved by the author
// ----------------------------------------------------------------
// cameraCapture.h - Header with the instructions for cameraCapture.cpp

#ifndef CAMCAP_H
#define CAMCAP_H

#include "/usr/include/opencv/cv.h"
#include "/usr/include/opencv/highgui.h"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <math.h>
#include <float.h>
#include <limits.h>
#include <time.h>
#include <ctype.h>
#include "eigenfaces.h"
#include "mixeraut.h"
class EigenFaces;
class cameraCapture
{

public:

    cameraCapture();
    int captureWindow();
    int detect_and_draw( IplImage* img );

private:

    QString pathSnap;
    EigenFaces *eigen;
    int numFacesUsed;
    void lookUpFace();
    CvCapture* capture;

};

#endif

// Master Thesis - StoneFace v.1.0 beta C by Jorge Garcia Bueno
// Universidad Carlos III de Madrid + University of Glasgow 2009
// All rights reserved by the author
// ----------------------------------------------------------------
// cameraCapture.cpp - Control the camera frames using OpenCV

#include "cameraCapture.h"

cameraCapture::cameraCapture()
{
    eigen = new EigenFaces(NULL);
    eigen->SetDB("./me.xml");
    numFacesUsed = 0;
}

int cameraCapture::captureWindow()
{

    capture = 0;
    // Images to capture the frame from video or camera or from file
    IplImage *frame, *frame_copy;



    // Find whether to detect the object from file or from camera.

        capture = cvCaptureFromCAM(-1);
    // Create a new named window with title: result
    cvNamedWindow( "result", 1 );
    //cvNamedWindow( "face", 0 );
    // Find if the capture is loaded successfully or not.

    // If loaded succesfully, then:
    if( capture )
    {
        // Capture from the camera.
        for(;;)
        {
            // Capture the frame and load it in IplImage
            if( !cvGrabFrame( capture ))
                break;
            frame = cvRetrieveFrame( capture );

            // If the frame does not exist, quit the loop
            if( !frame )
                break;

            // Allocate framecopy as the same size of the frame
            if( !frame_copy )
                frame_copy = cvCreateImage( cvSize(frame->width,frame->height),
                                            IPL_DEPTH_8U, frame->nChannels );


            // Check the origin of image. If top left, copy the image frame to frame_copy.
            //if( frame->origin == IPL_ORIGIN_TL )
                //cvCopy( frame, frame_copy, 0 );
            // Else flip and copy the image
            //else
                //cvFlip( frame, frame_copy, 0 );

            // Call the function to detect and draw the face
            detect_and_draw( frame );

            // Wait for a while before proceeding to the next frame
            if( cvWaitKey( 10 ) >= 0 )
                break;
        }

        // Release the images, and capture memory
        cvReleaseImage( &frame_copy );
        cvReleaseCapture( &capture );
    }
    else
    {
        fprintf(stderr, "No camera detected!\n");
    }

    // Destroy the window previously created with filename: "result"
    cvDestroyWindow("result");

    // return 0 to indicate successfull execution of the program
    return 0;
}

// Function to detect and draw any faces that is present in an image
int cameraCapture::detect_and_draw( IplImage* img )
{
    int scale = 1;
    QString pathSnap;
    // Creation of enough memory for the next computations
    static CvMemStorage* storage = 0;
    IplImage* imageface;

    // Creation of a new Haar classifier
    static CvHaarClassifierCascade* cascade = 0;
    // Load the HaarClassifierCascade
    cascade = (CvHaarClassifierCascade*)cvLoad( "./cascades/haarcascade_frontalface_alt.xml", 0, 0, 0 );

    // Check whether the cascade has loaded successfully. Else report and error and quit
    if( !cascade )
    {
        fprintf( stderr, "ERROR: Could not load classifier cascade\n" );

    }

    // Create a new image based on the input image
    IplImage* temp = cvCreateImage( cvSize(img->width,img->height), 8, 3 );

    // Create two points to represent the face locations
    CvPoint pt1, pt2;
    int i;
    // Allocate the memory storage
    storage = cvCreateMemStorage(0);

    // Clear the memory storage which was used before
    //cvClearMemStorage( storage );

    // Find whether the cascade is loaded, to find the faces. If yes, then:
    if( cascade )
    {

        // There can be more than one face in an image. So create a growable sequence of faces.
        // Detect the objects and store them in the sequence
        CvSeq* faces = cvHaarDetectObjects( img, cascade, storage,
                                            1.1, 2, CV_HAAR_DO_CANNY_PRUNING,
                                            cvSize(40, 40) );
        // Loop the number of faces found.
        for( i = 0; i < (faces ? faces->total : 0); i++ )
        {
           // Create a new rectangle for drawing the face
            CvRect* r = (CvRect*)cvGetSeqElem( faces, i );

            // Find the dimensions of the face,and scale it if necessary
            pt1.x = (r->x)*scale;
            pt2.x = (r->x+r->width )*scale;
            pt1.y = (r->y)*scale;
            pt2.y = (r->y+r->height)*scale;

            // Draw the rectangle in the input image
            cvRectangle( img, pt1, pt2, CV_RGB(255,0,255), 3, 8, 0 );
            cvSetImageROI(img,*r);

            //cvCopy(img,imageface);

            pathSnap = QString("./faces/face%1.jpg").arg( numFacesUsed++);

            // if the size is small, we discard the image
            if ( (img->height > 60) && (img->width > 60))
            {
                imageface = cvCreateImage( cvSize(180, 180), img->depth, img->nChannels );
                cvResize(img,imageface);
                cvSaveImage(pathSnap.toLatin1().data(),imageface);
            }
            else
            {
                numFacesUsed--;
            }

            cvResetImageROI(img);

            //lookUpFace();
            //imageface = cvLoadImage(pathSnap.toLatin1().data());
          if ( numFacesUsed == 80)
            {
                QStringList qsl;
                for (int io = 0;io<80;io++)
                    qsl << QString("./faces/face%1.jpg").arg( io);

                MixerAutomatic *mix =  new MixerAutomatic(NULL, NULL);
                mix->setFilesList(qsl);
                mix->Start(QString("average.jpg"),180,180,false);

                // Release the images, and capture memory
                //cvReleaseImage( &temp );
                //cvReleaseImage( &img );
                //cvReleaseImage( &imageface );
                cvDestroyWindow("result");
                return(0);
            }

        }
    }

    // Show the image in the window named "result"
    cvShowImage( "result", img );
    cvShowImage("face",imageface );
    // Release the temp image created.
    cvReleaseImage( &temp );
}


void cameraCapture::lookUpFace()
{
    eigen->Recognise(pathSnap.toLatin1().data(), true);
}



// Master Thesis - StoneFace v.1.0 beta C by Jorge Garcia Bueno
// Universidad Carlos III de Madrid + University of Glasgow 2009
// All rights reserved by the author
// -------------------------------------------------------------
// detector.h - Header with the instructions for detector.cpp

#ifndef DETECTOR_H
#define DETECTOR_H


// Include header files
#include "/usr/include/opencv/cv.h"
#include "/usr/include/opencv/highgui.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <math.h>
#include <float.h>
#include <limits.h>
#include <time.h>
#include <ctype.h>

class Detector
{

private:

    // Image to be treated
    IplImage* image;
    void ImagePreProcessForDetection();
public:

    // Function to detect the elements that returns the coordinates of two points
    void DetectAndDraw(char, int* , int* , int* , int* );
    // Constructor of the object
    Detector(IplImage*);
};

#endif

// Master Thesis - StoneFace v.1.0 beta C by Jorge Garcia Bueno
// Universidad Carlos III de Madrid + University of Glasgow 2009
// All rights reserved by the author
// --------------------------------------------------------------
// detector.cpp - Implementation of Haar-Cascades to detect and return position of face features

#include "detector.h"


Detector::Detector(IplImage* ima)
{

    image = ima;
}

void Detector::ImagePreProcessForDetection()
{
    cvEqualizeHist( image, image );
}

void Detector::DetectAndDraw(char cascadeType, int* px, int* py, int* imHeight, int* imWidth )
{

    // Creation of enough memory for the next computations
    static CvMemStorage* storage = 0;

    // Creation of a new Haar classifier
    static CvHaarClassifierCascade* cascade = 0;

    // Scale for improvement pourpose
    int scale = 1;

    // Path of the cascade file
    char* cascadeName = NULL;

    // Variables for the nose selection - The most centered
    int pxSelected = 0,pySelected = 0,hSelected = 0,wSelected = 0;
    // Select the path of the cascade

    switch (cascadeType)
    {
        case 'F':   // Face location
        {                                   //haarcascade_frontalface_alt_tree
            cascadeName = (char*)"./cascades/haarcascade_frontalface_alt2.xml";
            break;
        }
        case 'R':   // Right eye location
        {
            cascadeName = (char*)"./cascades/right_eye.xml";
            break;
        }
        case 'L':   // Left eye location
        {
            cascadeName = (char*)"./cascades/left_eye.xml";
            break;
        }
        case 'M':   // Mouth location
        {
            cascadeName = (char*)"./cascades/mouth.xml";
            break;
        }
        case 'N':   // Nose location
        {
            cascadeName = (char*)"./cascades/nose.xml";
            break;
        }

        case 'E':   // Both eyes location
        {
            cascadeName = (char*)"./cascades/eyes.xml";
            break;
        }
    }
    // Load the cascade that corresponds to the detection are we looking for
    cascade = (CvHaarClassifierCascade*)cvLoad( cascadeName, 0, 0, 0 );

    // Check whether the cascade has loaded successfully. Else report and error and quit
    if( !cascade )
    {
        fprintf( stderr, "ERROR: Could not load classifier cascade\n" );
        return;
    }


    // Allocate the memory storage
    storage = cvCreateMemStorage(0);

    // Clear the memory storage which was used before
    cvClearMemStorage( storage );

    // Find whether the cascade is loaded, to find the faces. If yes, then:
    if( cascade )
    {

        // There can be more than one face in an image. So create a growable sequence of faces.
        // Detect the objects and store them in the sequence
        CvSeq* faces = cvHaarDetectObjects( image, cascade, storage,
                                            1.1, 5, CV_HAAR_DO_CANNY_PRUNING);//,
                                            //cvSize(40, 40) );


        // Delete previous values and find the best one
        *px = 0; *py = 0; *imWidth = 0; *imHeight = 0; pxSelected = image->width*scale;
        // Loop the number of faces found.
        for( int i = 0; i < (faces ? faces->total : 0); i++ )
        {
            // Create a new rectangle to pick up the coordinates
            CvRect* r = (CvRect*)cvGetSeqElem( faces, i );

            // Get the dimensions of the face and scale it if necessary
            *px = r->x*scale;
            *py = r->y*scale;
            *imWidth = r->width*scale;
            *imHeight = r->height*scale;
            // Selection of the most centered nose
            if (cascadeType == 'N')
                {
                    if (abs(*px - image->width*scale/2) <=   abs(pxSelected - image->width*scale/2))
                        {
                            //fprintf(stderr,"old x %d new x %d\n",pxSelected,*px);
                            pxSelected = *px;
                            pySelected = *py;
                            wSelected = *imWidth;
                            hSelected = *imHeight;
                        }
                }
            //fprintf(stderr,"Search of: %c [%d] px=%d py=%d wid=%d hei=%d\n", cascadeType, i,*px,*py,*imW
idth, *imHeight);
        }
    }

    // Check if the case if nose selection to update with the most centered
    if (cascadeType == 'N')
```

```
        {
            *px = pxSelected; *py = pySelected; *imWidth = wSelected; *imHeight = hSelected;
        }
    // Release the image
    cvReleaseImage(&image);

}

// Master Thesis - StoneFace v.1.0 beta C by Jorge Garcia Bueno
// Universidad Carlos III de Madrid + University of Glasgow 2009
// All rights reserved by the author
// -----------------------------------------------------------------
// eigenfaces.h - Header with the instructions for eigenfaces.cpp

#ifndef EIGEN_H
#define EIGEN_H

#include <stdio.h>
#include <string.h>
#include <QtGui>
#include <QPixmap>
#include <QGraphicsView>
#include <QListView>
#include <QDialog>
#include <QPushButton>
#include <QDesktopWidget>
#include "/usr/include/opencv/cv.h"
#include "/usr/include/opencv/cvaux.h"
#include "/usr/include/opencv/highgui.h"
#include "searchwidget.h"
#include "detector.h"

class SearchWidget;

class EigenFaces : public QWidget
{
    Q_OBJECT


private:

    // Array of face images
    IplImage ** faceImgArr;
    // Array of person numbers
    CvMat    *  personNumTruthMat;
    // Number of training images
    int nTrainFaces;
    // Number of eigenvalues
    int nEigens;
    // Average image
    IplImage * pAvgTrainImg;
    // Eigenvectors
    IplImage ** eigenVectArr;
    // Eigenvalues
    CvMat * eigenValMat;
    // Projected training faces
    CvMat * projectedTrainFaceMat;

    // Parent widget
    QWidget *searchwidget;
    // Database file path
    char *faceDB;
    QString *faceDBS;
    QString *faceDBinfo;
    void GetInfoDB();
    void DoPCA();
    void StoreTrainingData();
    int  LoadTrainingData(CvMat ** pTrainPersonNumMat);
    int  FindNearestNeighbor(float * projectedTestFace, double *distance, int *percentage);
    int  LoadFaceImgArray(char * filename);
    void Learn();
    void SaveJustTheFace(QString pathImg,QString pathResized);
    void CreateRecogniseList(char* pathOfImage, bool);

    QString GetInfo(int);

public:

    EigenFaces(QWidget *parent);
    void SetDB(char *);
    void Recognise(char* pathOfImage, bool);
signals:

    void InfoDB(QString , int , int , int );
    void NearestImage(QString);
    void Progress(int);
    void InfoNearest(QString);
};


#endif

// Master Thesis - StoneFace v.1.0 beta C by Jorge Garcia Bueno
// Universidad Carlos III de Madrid + University of Glasgow 2009
// All rights reserved by the author
// --------------------------------------------------------------
// eigenfaces.cpp - Implementation of the eigenfaces generation including face-database
// Some parts are extracted from http://svn.assembla.com/svn/face/mfcBitirme

#include "eigenfaces.h"


EigenFaces::EigenFaces(QWidget *parent)
{

    searchwidget = parent;
    faceImgArr = 0;

     // Array of person numbers
    personNumTruthMat = 0;
    // Number of training images
    nTrainFaces = 0 ;
    // Number of eigenvalues
    nEigens = 0 ;
    // Average of image
    pAvgTrainImg = 0 ;
    // Eigenvectors
    eigenVectArr = 0 ;
    // Eigenvalues
    eigenValMat = 0;
    // Projected training faces
    projectedTrainFaceMat=0 ;


}

void EigenFaces::GetInfoDB()
{

    //int nEigen = 0;
    int nFaces = 0;
    char line[30];
    FILE* f=fopen(faceDB,"r");
    if (!f) {
        fprintf(stderr,"Can't open database to get the info: %s\n", faceDB);
        exit(1);
    }


    /*
    <?xml version="1.0"?>
    <opencv_storage>
    <nEigens>41</nEigens>
    <nTrainFaces>42</nTrainFaces>
    */
    int x = 0;
    x = fscanf(f,"%s\n",line);
    x = fscanf(f,"%s\n",line);
    x = fscanf(f,"%s\n",line);
    x = fscanf(f,"%s\n",line);

    fclose(f);


    QString *g = new QString(line);
    QString *gg = new QString(g->mid(9));
    QString *ggg = new QString(gg->left(gg->length() - 10));

    nFaces = ggg->toInt();
    //printf("nFaces: %d string %s\n", nFaces, ggg->toLatin1().data());
    QFileInfo *fileInfo = new QFileInfo(QString(faceDB));



    emit InfoDB(fileInfo->created().toString(), nFaces, 190, 285);
}

void EigenFaces::SetDB(char *fdb)
{
    faceDB = fdb;
    faceDBS = new QString(fdb);
    faceDBinfo = new QString(fdb);
    faceDBinfo->chop(4);
    faceDBinfo->append(QString("-info.txt"));
    GetInfoDB();

}

void EigenFaces::SaveJustTheFace(QString pathImg,QString pathResized)
{

    IplImage *ImageFace = 0;
    Detector *dFace;
    int px,py,imHeight,imWidth;

    QImage *qimg = new QImage(pathImg);


    // Extraction of the face of the picture...

    ImageFace = cvLoadImage(pathImg.toLatin1().data());

    // Creation of the detector and obtaining of the coordinates for the face
    dFace =  new Detector(ImageFace);
    dFace->DetectAndDraw('F',&px,&py,&imHeight,&imWidth);
    printf(" px = %d py = %d h = %d w = %d\n", px,py,imHeight,imWidth);
    // Now we just want the face scaled with the coordinates changed and work with result

    QImage result = qimg->copy(px,py,imHeight,imWidth).scaled(90,90,Qt::IgnoreAspectRatio,Qt::SmoothTransf
ormation);
    result.save(pathResized);


}

// Not only creates a list with the image for being recognized but also detects the face in big images
// and extract it from the image for being recognized later... :D
void EigenFaces::CreateRecogniseList(char* pathOfImage, bool aloneInTheImage)
{
    FILE * f = NULL;

    QString pathresized("/tmp/resized.jpg");
    //pathresized = QString(pathOfImage) + QString("-resized.jpg") ;

    if (aloneInTheImage == false)
    {
        // Extract the face from the image and resize
        SaveJustTheFace(pathOfImage, pathresized);
    }
    else
    {
        QImage *resizedImg = new QImage(pathOfImage);
        resizedImg = new QImage(resizedImg->scaled(90,90,Qt::IgnoreAspectRatio,Qt::SmoothTransformation));
        resizedImg->save(pathresized);
    }
    // open the input file
    if( !(f = fopen((char*)"_recog.tmp", "w")) )
    {
        fprintf(stderr, "Can\'t write face list file %s\n Space problems? Rights problems?\n", (char*)"_re
cog.tmp");
        exit( 0);
    }

    int ret;

    ret = fprintf(f,"1 %s", pathresized.toLatin1().data());

    fclose(f);
}



//////////////////////////////////
// recognize()
//
void EigenFaces::Recognise(char* pathOfImage, bool aloneInTheImage)
{
    int i, nTestFaces  = 0;         // the number of test images
    CvMat * trainPersonNumMat = 0;  // the person numbers during training
    float * projectedTestFace = 0;
    double distanceNearest;
    int per = 0;                    // Percentage of similarity

    // Call the wrapper function in charge of create temporal file recognise.txt
    CreateRecogniseList(pathOfImage, aloneInTheImage);

    // load test images and ground truth for person number
    nTestFaces = LoadFaceImgArray((char*)"_recog.tmp");
    //printf("%d test faces loaded\n", nTestFaces);

    // load the saved training data
    if( !LoadTrainingData( &trainPersonNumMat ) ) return;

    // project the test images onto the PCA subspace
    projectedTestFace = (float *)cvAlloc( nEigens*sizeof(float) );
    for(i=0; i<nTestFaces; i++)
    {
        int iNearest, nearest, truth;

        // project the test image onto the PCA subspace
        cvEigenDecomposite(
            faceImgArr[i],
            nEigens,
            eigenVectArr,
            0, 0,
            pAvgTrainImg,
            projectedTestFace);

        iNearest = FindNearestNeighbor(projectedTestFace,&distanceNearest, &per);
        truth    = personNumTruthMat->data.i[i];
        nearest  = trainPersonNumMat->data.i[iNearest];
        //if (nearest != truth) printf("·· FAIL! :: ");
        //printf("N/T = %d/%d [%d%%] - distance = %f\n", nearest,truth,per,distanceNearest);

        FILE* f=fopen(faceDBinfo->toLatin1().data(),"r");
        int z = 0, x = 0;
        char fileName[128];

        for (int iRow=0;iRow<iNearest+1;iRow++)
            x = fscanf(f,"%d %s\n",&z,&fileName[0]);

        fclose (f);

        printf("FileName: %s\n",fileName);
        emit (NearestImage(fileName));

        //emit(InfoNearest(GetInfo(iNearest)));
    }
}

QString EigenFaces::GetInfo(int position)
{

    int k = 0;
    QString line;
    QString pathInfoSubjects(*faceDBinfo);
    pathInfoSubjects.chop(9);
    pathInfoSubjects.append(QString("-extra.txt"));

    QFile file(pathInfoSubjects);

    if (!file.open(QIODevice::ReadOnly | QIODevice::Text))
        return QString("Error loading info!");

    QTextStream in(&file);
    while (k < position+1)
    {
        line = in.readLine();
        k++;
    }
    printf("captured: %s\n",line.toLatin1().data());
    QStringList list1 = line.split(" ");
    QString infoFaceL("");
    infoFaceL += "Information of the matched face:\n";
    infoFaceL += "Name: " + QString(list1[0].replace(QString("_"),QString(" "))) + " " + QString(list1[1].
replace(QString("_"),QString(" "))) + "\n";
    infoFaceL += "Age: " + QString(list1[2]) + "\n";
    infoFaceL += "Gender: " + QString(list1[3]) + "\n";
    infoFaceL += "Info: " + QString(list1[4].replace(QString("_"),QString(" "))) + "\n";
    infoFaceL += "Web: " + QString(list1[5]);
    return  infoFaceL;

}

//////////////////////////////////
// loadTrainingData()
//
int EigenFaces::LoadTrainingData(CvMat ** pTrainPersonNumMat)
{
    CvFileStorage * fileStorage;
    int i;

    // create a file-storage interface

    fileStorage = cvOpenFileStorage( faceDBS->toLatin1().data(), 0, CV_STORAGE_READ );
    if( !fileStorage )
    {
        fprintf(stderr, "Can't open database %s to start training data\n",faceDB);
        return 0;
    }

    nEigens = cvReadIntByName(fileStorage, 0, "nEigens", 0);
    nTrainFaces = cvReadIntByName(fileStorage, 0, "nTrainFaces", 0);
    *pTrainPersonNumMat = (CvMat *)cvReadByName(fileStorage, 0, "trainPersonNumMat", 0);
    eigenValMat  = (CvMat *)cvReadByName(fileStorage, 0, "eigenValMat", 0);
    projectedTrainFaceMat = (CvMat *)cvReadByName(fileStorage, 0, "projectedTrainFaceMat", 0);
    pAvgTrainImg = (IplImage *)cvReadByName(fileStorage, 0, "avgTrainImg", 0);
    eigenVectArr = (IplImage **)cvAlloc(nTrainFaces*sizeof(IplImage *));
    for(i=0; i<nEigens; i++)
    {
        char varname[200];
        sprintf( varname, "eigenVect_%d", i );
        eigenVectArr[i] = (IplImage *)cvReadByName(fileStorage, 0, varname, 0);
    }

    // release the file-storage interface
    cvReleaseFileStorage( &fileStorage );

    return 1;
}


//////////////////////////////////
// storeTrainingData()
//
void EigenFaces::StoreTrainingData()
{
    CvFileStorage * fileStorage;
    int i;

    // create a file-storage interface
    fileStorage = cvOpenFileStorage( "facedata.xml", 0, CV_STORAGE_WRITE );

    // store all the data
    cvWriteInt( fileStorage, "nEigens", nEigens );
    cvWriteInt( fileStorage, "nTrainFaces", nTrainFaces );
    cvWrite(fileStorage, "trainPersonNumMat", personNumTruthMat, cvAttrList(0,0));
    cvWrite(fileStorage, "eigenValMat", eigenValMat, cvAttrList(0,0));
    cvWrite(fileStorage, "projectedTrainFaceMat", projectedTrainFaceMat, cvAttrList(0,0));
    cvWrite(fileStorage, "avgTrainImg", pAvgTrainImg, cvAttrList(0,0));
    for(i=0; i<nEigens; i++)
    {
        char varname[200];
        sprintf( varname, "eigenVect_%d", i );
        cvWrite(fileStorage, varname, eigenVectArr[i], cvAttrList(0,0));
    }

    // release the file-storage interface
    cvReleaseFileStorage( &fileStorage );
}


//////////////////////////////////
// findNearestNeighbor()
//
int EigenFaces::FindNearestNeighbor(float * projectedTestFace, double *distance, int *percentage)
{
    //double leastDistSq = 1e12;
    double leastDistSq = DBL_MAX,maxDist=0;
    int i, iTrain, iNearest = 0;

    for(iTrain=0; iTrain<nTrainFaces; iTrain++)
    {
        double distSq=0;
        emit Progress((int)(100*iTrain/nTrainFaces));
        for(i=0; i<nEigens; i++)
        {
            float d_i =
                projectedTestFace[i] -
                projectedTrainFaceMat->data.fl[iTrain*nEigens + i];
                distSq += d_i*d_i / eigenValMat->data.fl[i];  // Mahalanobis

            //distSq += d_i*d_i; // Euclidean
        }
        //printf("Distance for number %2i is %f\n", iTrain, distSq);
        if(distSq < leastDistSq)
        {
            leastDistSq = distSq;
            iNearest = iTrain;
            *distance = distSq;
        }
        // Maximum distance for percentage approximation
        if (distSq>maxDist) maxDist = distSq;
    }
    *percentage = (int)(100 *(maxDist - leastDistSq)/maxDist);
    return iNearest;
}


//////////////////////////////////
// doPCA()
//
void EigenFaces::DoPCA()
{
    int i;
    CvTermCriteria calcLimit;
    CvSize faceImgSize;

    // set the number of eigenvalues to use
    nEigens = nTrainFaces-1;

    // allocate the eigenvector images
    faceImgSize.width  = faceImgArr[0]->width;
    faceImgSize.height = faceImgArr[0]->height;
    eigenVectArr = (IplImage**)cvAlloc(sizeof(IplImage*) * nEigens);
    for(i=0; i<nEigens; i++)
        eigenVectArr[i] = cvCreateImage(faceImgSize, IPL_DEPTH_32F, 1);

    // allocate the eigenvalue array
    eigenValMat = cvCreateMat( 1, nEigens, CV_32FC1 );

    // allocate the averaged image
    pAvgTrainImg = cvCreateImage(faceImgSize, IPL_DEPTH_32F, 1);

    // set the PCA termination criterion
    calcLimit = cvTermCriteria( CV_TERMCRIT_ITER, nEigens, 1);
    printf("PCA done!\n");
    // compute average image, eigenvalues, and eigenvectors
    cvCalcEigenObjects(
        nTrainFaces,
        (void*)faceImgArr,
        (void*)eigenVectArr,
        CV_EIGOBJ_NO_CALLBACK,
        0,
        0,
        &calcLimit,
        pAvgTrainImg,
        eigenValMat->data.fl);

    cvNormalize(eigenValMat, eigenValMat, 1, 0, CV_L1, 0);
}


//////////////////////////////////
// loadFaceImgArray()
//
int EigenFaces::LoadFaceImgArray(char * filename)
{
    FILE * imgListFile = 0;
    char imgFilename[512];
    int iFace, nFaces=0;
    int x = 0;

    // open the input file
    if( !(imgListFile = fopen(filename, "r")) )
    {
        fprintf(stderr, "Can\'t open file %s\n", filename);
        return 0;
    }

    // count the number of faces
    while( fgets(imgFilename, 512, imgListFile) ) ++nFaces;
    rewind(imgListFile);

    // allocate the face-image array and person number matrix
    faceImgArr        = (IplImage **)cvAlloc( nFaces*sizeof(IplImage *) );
    personNumTruthMat = cvCreateMat( 1, nFaces, CV_32SC1 );

    // store the face images in an array
    for(iFace=0; iFace<nFaces; iFace++)
    {
        // read person number and name of image file
         x = fscanf(imgListFile,
            "%d %s", personNumTruthMat->data.i+iFace, imgFilename);

        // load the face image
        faceImgArr[iFace] = cvLoadImage(imgFilename, CV_LOAD_IMAGE_GRAYSCALE);

        if( !faceImgArr[iFace] )
        {
            fprintf(stderr, "Can\'t load image from %s\n", imgFilename);
            return 0;
        }
    }

    fclose(imgListFile);

    return nFaces;
}

void EigenFaces::Learn()
{
    int i, offset;

    // load training data
    nTrainFaces = LoadFaceImgArray((char*)"learn.txt");
    if( nTrainFaces < 2 )
    {
        fprintf(stderr,
                "Need 2 or more training faces\n"
                "Input file contains only %d\n", nTrainFaces);
        return;
    }

    // do PCA on the training faces

    DoPCA();
    printf("PCA done!\n");
    // project the training images onto the PCA subspace
    projectedTrainFaceMat = cvCreateMat( nTrainFaces, nEigens, CV_32FC1 );
    offset = projectedTrainFaceMat->step / sizeof(float);
    for(i=0; i<nTrainFaces; i++)
    {
        //int offset = i * nEigens;
        cvEigenDecomposite(
            faceImgArr[i],
            nEigens,
            eigenVectArr,
            0, 0,
            pAvgTrainImg,
            //projectedTrainFaceMat->data.fl + i*nEigens);
            projectedTrainFaceMat->data.fl + i*offset);
    }

    // store the recognition data as an xml file
    StoreTrainingData();
}

// Master Thesis - StoneFace v.1.0 beta C by Jorge Garcia Bueno
// Universidad Carlos III de Madrid + University of Glasgow 2009
// All rights reserved by the author
// ---------------------------------------------------------------
// graphWidget.h - Header with the instructions for graphWidget.cpp

#ifndef GRAPHWIDGET_H
#define GRAPHWIDGET_H

#include <QtGui/QGraphicsView>

#include "node.h"
#include "detector.h"



class Node;

class GraphWidget : public QGraphicsView
{
    Q_OBJECT

public:
    GraphWidget();

    void itemMoved();
    QImage currentImage();
    void nodeMoved(int nodeNumber, qreal newX,qreal newY);
    void savePicture(const QString &fileName);
    int loadPicture(const QString &fileName, char* origMesh, char* destMesh,bool autodetection);
    int updatePicture(bool showInWindow);
    void downloadAddMesh(QList<qreal>*);
    void printEdges();
    void removeEdges();
    void nodeClicked(int nodeNumber);
    void refreshPoints();
    int detectElements(bool stretch);
    void helpImageVisible(bool);
protected:
    void keyPressEvent(QKeyEvent *event);
    void timerEvent(QTimerEvent *event);
    void wheelEvent(QWheelEvent *event);
    void drawBackground(QPainter *painter, QImage &face);

    void scaleView(qreal scaleFactor);
    int aNodes[86][2];
    int aNodes2[86][2];
    int hNodes[86][2];
    int aEdges[68][2];
    int aTri[166][3];
    int nTri,nNodes;
    bool inside, flag;

    double A[3][3];
    double in[3][3], alpha, beta, gamma;
    double Apx,Apy,Bpx,Bpy,Cpx,Cpy,Xp,Yp;
    double detInv;//n is the determinant of A

private:
    QPixmap *fa1,*fa2,*faRef;
    long lineLength(int A1, int B1, int A2, int B2);
    double areaOfTriangle(int Ax, int Ay, int Bx, int By, int Cx, int Cy);

    int timerId;
    int lastClicked;
    bool showEdges;
    bool insideTriangle(int Ax, int Ay, int Bx, int By, int Cx, int Cy, int Px, int Py);
    QGraphicsPixmapItem *face1,*faceReference;
    QImage *imf1,*imf2;
    QGraphicsScene *superScene;
    Node** hnode;
    QGraphicsLineItem** qline;
    QString path;
    void showWarpedWindow(QImage* &pic);

    int centerGroupX;
    int centerGroupY;

    bool showlines;
public slots:

    // To manage communications with mainwindow

    // For one node
    void displaceNodeLeft();
    void displaceNodeRight();
    void displaceNodeUp();
    void displaceNodeDown();

    // For a group of nodes
    void displaceGroupLeft();
    void displaceGroupRight();
    void displaceGroupUp();
    void displaceGroupDown();

    void turnGroupLeft();
    void turnGroupRight();

    void increaseGroupDistance();
    void decreaseGroupDistance();

    // For the whole face
    void displaceAllLeft();
    void displaceAllRight();
    void displaceAllUp();
    void displaceAllDown();


};

#endif

// Master Thesis - StoneFace v.1.0 beta C by Jorge Garcia Bueno
// Universidad Carlos III de Madrid + University of Glasgow 2009
// All rights reserved by the author
// -------------------------------------------------------------
// graphWidget.cpp - Main object. Controls the warped image and the interface to control the contained fac
e

#include "graphwidget.h"


#include <QDebug>
#include <QGraphicsScene>
#include <QWheelEvent>
#include <QtGui>
#include <QPixmap>
#include <QGraphicsView>
#include <math.h>
#include <QPushButton>
#include <QSplashScreen>
```

```
GraphWidget::GraphWidget()
    : timerId(0)
{


    QGraphicsScene *scene = new QGraphicsScene(this);
    scene = new QGraphicsScene(this);
    superScene = scene;
    scene->setItemIndexMethod(QGraphicsScene::NoIndex);
    scene->setSceneRect(0,0, 760, 570);
    setScene(scene);
    //setViewportUpdateMode(BoundingRectViewportUpdate);
    //setCacheMode(CacheBackground);
    //setRenderHint(QPainter::Antialiasing);
    //setTransformationAnchor(AnchorUnderMouse);
    //setResizeAnchor(AnchorViewCenter);
    showEdges = true;
    // restart the last node clicked for colors pourpose
    lastClicked = -1;

    centerGroupX = 0;
    centerGroupY = 0;

    showlines = false;
}

void GraphWidget::itemMoved()
{
    if (!timerId)
        timerId = startTimer(1 / 25);
}

void GraphWidget::nodeMoved(int nodeNumber, qreal newX,qreal newY)
{
    // Refresh the picture with the new image

    aNodes[nodeNumber][0] = (int)newX;
    aNodes[nodeNumber][1] = (int)newY;


}


// Interaction between the user and the nodes of the mesh

void GraphWidget::keyPressEvent(QKeyEvent *event)
{

    // Creation of a list with the nodes of the Scene to interact with

    QList<Node *> nodes;
    foreach (QGraphicsItem *item, scene()->items())
        if (Node *node = qgraphicsitem_cast<Node *>(item))
            // Ignore the help nodes of the right picture (only for help pourpose)
            if (node->helpNode == false) nodes << node;


    switch (event->key())
    {

    // Key Up -> move the last clicked node 2 pixels up
    case Qt::Key_Up:
        {
            displaceNodeUp();
            break;
        }

    // Key Down -> move the last clicked node 2 pixels down
    case Qt::Key_Down:
        {
            displaceNodeDown();
            break;
        }

    // Key Left -> move the last clicked node 2 pixels left
    case Qt::Key_Left:
        {
            displaceNodeLeft();
            break;
        }

    // Key Right -> move the last clicked node 2 pixels right
    case Qt::Key_Right:
        {
            displaceNodeRight();
            break;
        }

    // Key A -> move the "color group" of last clicked node 2 pixels left
    case Qt::Key_A:
        {
            displaceGroupLeft();
            break;
        }

    // Key A -> move the "color group" of last clicked node 2 pixels right
    case Qt::Key_D:
        {
            displaceGroupRight();
            break;
        }

    // Key S -> move the "color group" of last clicked node 2 pixels down
    case Qt::Key_S:
        {
            displaceGroupDown();
            break;
        }

    // Key W -> move the "color group" of last clicked node 2 pixels up
    case Qt::Key_W:
        {
            displaceGroupUp();
            break;
        }

    // Key F -> move all nodes 2 pixels left
    case Qt::Key_F:
        {
            displaceAllLeft();
            break;
        }

    // Key H -> move all nodes 2 pixels right
    case Qt::Key_H:
        {
            displaceAllRight();
            break;
        }

    // Key G -> move all nodes 2 pixels down
    case Qt::Key_G:
        {
            displaceAllDown();
            break;
        }

    // Key T -> move all nodes 2 pixels up
    case Qt::Key_T:
        {
            displaceAllUp();
            break;
        }

    // Key + -> Increase the dispersion of the members of a group pixels
    case Qt::Key_Plus:
        {
            increaseGroupDistance();
            break;
        }

    // Key - -> Decrease the dispersion of the members of a group pixels
    case Qt::Key_Minus:
        {
            decreaseGroupDistance();
            break;
        }

    case Qt::Key_P:

    case Qt::Key_M:
        foreach (Node *node, nodes)
            fprintf(stderr,"%d %d %c\n",(int)node->posX(),(int)node->posY(),node->getColor());
        break;

    default:
        QGraphicsView::keyPressEvent(event);
    }

}

void GraphWidget::removeEdges()
{

    showlines = false;

    foreach (QGraphicsItem *item, scene()->items()) {
    // type 7 :: background pictures
    // type 3 :: squares
    // type 65537 :: all
        if (item->type() == 6)
            item->hide();

    }
}


void GraphWidget::timerEvent(QTimerEvent *event)
{
    Q_UNUSED(event);

    QList<Node *> nodes;

    foreach (QGraphicsItem *item, scene()->items())
    {
        if (Node *node = qgraphicsitem_cast<Node *>(item))
            if (node->helpNode == false)
                nodes << node;
    }

    foreach (Node *node, nodes)
        node->calculateForces();

    bool itemsMoved = false;
    foreach (Node *node, nodes)
    {
        if (node->advance())
            itemsMoved = true;
    }
    if (!itemsMoved)
    {
        killTimer(timerId);
        timerId = 0;
    }
}

QImage GraphWidget::currentImage()
{
    return *imf2;
}


void GraphWidget::showWarpedWindow(QImage* &pic)
{
     // Creation of a new form to show the warped image

    pic->save("temp-warp.jpg");
    IplImage* tempx = cvLoadImage("temp-warp.jpg", -1);
    cvNamedWindow("Temporal warped image", 1);
    cvShowImage( "Temporal warped image", tempx);

}


int GraphWidget::loadPicture(const QString &fileName, char* origMesh, char* destMesh,bool autodetection)
{

    // ~~~~~~~~~~~~~~ LOAD AND SET THE PIXMAPS AND PICTURES ~~~~~~~~~~~~~~

    // fa1 contains the working picture
    // imf2 contains the information of fa1 transformed in QPixmap
    // face1 is a PixmapItem ready to be inserted into the scene

    path = fileName;
    fa1 = new QPixmap(fileName);
    imf1 = new QImage(fa1->toImage());

    // Resize the pictures -- probably not necessary because input is normallized

    if ((fa1->width() != 380) || (fa1->height() != 570) )
    {

        *fa1 = fa1->scaled(380,570,Qt::IgnoreAspectRatio,Qt::SmoothTransformation);
        *imf1 = imf1->scaled(380,570,Qt::IgnoreAspectRatio,Qt::SmoothTransformation);

    }


    face1 = new QGraphicsPixmapItem(*fa1);

    // fa2 contains the base picture (and final warped solution)
    // imf2 contains the information of fa2 transformed in QPixmap

    fa2 = new QPixmap(fileName);
    imf2 = new QImage(fa2->toImage());

    // Resize the pictures -- probably not necessary because input is normallized

    if ((fa2->width() != 380) || (fa2->height() != 570) )
    {

        *fa2 = fa2->scaled(380,570,Qt::IgnoreAspectRatio,Qt::SmoothTransformation);
        *imf2 = imf2->scaled(380,570,Qt::IgnoreAspectRatio,Qt::SmoothTransformation);

    }

    // Creation and location of the picture with the points reference
```

```
    faRef = new QPixmap("avgPoints.bmp");
    faceReference = new QGraphicsPixmapItem(*faRef);

    // Addition of the items into the scene

    superScene->addItem(face1);
    superScene->addItem(faceReference);
    face1->setPos(0,0);
    faceReference->setPos(380,0);


    // ~~~~~~~~~~~~~~ LOAD MESH FILES, EDGES AND TRIANGLES's FILES ~~~~~~~~~~~~~~


    // By default we position all the nodes in the final destination.
    // After the recognition the positions will change

    // Load the original mesh and check if there is any input error

    FILE* f=fopen(origMesh,"r");
    FILE* g=fopen("posHelpNodes.dat","r");

    if (!f)
    {

        QMessageBox::warning(this, tr("Warp Faces"),
                             tr("Original mesh file %1 is missing!\nCannot read the file.")
                             .arg(origMesh));

        //fprintf(stderr,"Original mesh: %s is missing!\n",origMesh);
        exit(1);
    }

    if (!g)
    {

        QMessageBox::warning(this, tr("Warp Faces"),
                             tr("Help mesh file posHelpNodes.dat is missing!\nCannot read the file.\n
") );


        //fprintf(stderr,"Helping mesh: posHelpNodes.dat is missing!\n");
        exit(1);

    }

    int x, y;
    char color;

    // Load the number of nodes
    int z = fscanf(f,"%d\n",&nNodes);
    int m = fscanf(g,"%d\n",&nNodes);

    // Creation of the array of nodes and
    // creation of the array of helping nodes

    Node** node=new Node*[nNodes];

    hnode = new Node*[nNodes];

    // Fill the array with the values of each line until the end
    for(int i=0; i<nNodes; i++)
    {

        // Scan and addition to the scene

        // First help nodes
        m = fscanf(g,"%d %d %c\n",&x,&y,&color);
        hnode[i]= new Node(this,i,'m',true);
        superScene->addItem(hnode[i]);
```

```
        hnode[i]->setPos(x+380,y);

        // Fill the array with the positions
        hNodes[i][0]=x+380;
        hNodes[i][1]=y;

        // Second working nodes
        z = fscanf(f,"%d %d %c\n",&x,&y,&color);
        node[i]= new Node(this,i,color,false);
        superScene->addItem(node[i]);
        node[i]->setPos(x,y);

        // Fill the array with the positions
        aNodes[i][0]=x;
        aNodes[i][1]=y;

        // Refresh the position of the set of nodes
        node[i]->calculateForces();
        hnode[i]->calculateForces();
    }

    // Close the files
    fclose(f);
    fclose(g);


    // Read the final mesh and check if there is any input error
    FILE* f3=fopen(destMesh,"r");
    if (!f3)
    {

        QMessageBox::warning(this, tr("Warp Faces"),
                                tr("Frontal mesh file %1 is missing!\nCannot read the file.")
                                .arg(destMesh));

        exit(-1);
    }

    int nNodes2;
    char c;

    // Load the number of nodes
    z = fscanf(f3,"%d\n",&nNodes2);

    // Fill the array with the values of each line until the end
    for (int k=0;k<nNodes2;k++)
        z = fscanf(f3,"%d %d %c\n",&aNodes2[k][0],&aNodes2[k][1], &c);

    // Close the file
    fclose(f3);

    // Read the points of the triangles and check if there is any input error
    FILE* f4=fopen("triangles.dat","r");

    if (!f4)
    {

        QMessageBox::warning(this, tr("Warp Faces"),
                                tr("Frontal mesh file triangles.dat is missing!\nCannot read the file.") );
        exit(-1);
    }

    // Load the number of triangles
    z = fscanf(f4,"%d\n",&nTri);

    // Lines that represent the triangles
    qline = new QGraphicsLineItem*[3*nTri];

    int yu=0;
    // Fill the array with the values of each line until the end
    for (int k=0;k<nTri;k++)
    {

        z = fscanf(f4,"%d %d %d\n",&aTri[k][0],&aTri[k][1],&aTri[k][2]);
        qline[yu]= new QGraphicsLineItem(aNodes[aTri[k][0]][0],aNodes[aTri[k][0]][1],aNodes[aTri[k][1]]
[0],aNodes[aTri[k][1]][1]);
        superScene->addItem(qline[yu]);yu ++;
        qline[yu]= new QGraphicsLineItem(aNodes[aTri[k][1]][0],aNodes[aTri[k][1]][1],aNodes[aTri[k][2]]
[0],aNodes[aTri[k][2]][1]);
        superScene->addItem(qline[yu]);yu ++;
        qline[yu]= new QGraphicsLineItem(aNodes[aTri[k][0]][0],aNodes[aTri[k][0]][1],aNodes[aTri[k][2]]
[0],aNodes[aTri[k][2]][1]);
        superScene->addItem(qline[yu]);yu ++;

    }

    // Close the file
    fclose(f4);

    // Add the lines

    // Scalation of the picture
    scale(1,1);

    // Properties of the window
    setMinimumSize(300, 300);

    refreshPoints();

    // Do we have to detect automatically the points of the destination mesh?
    if (autodetection == true)
    {
        if (detectElements(false) == 0)

            // Refresh the points
            refreshPoints();

            // There was a problems while detecting the points...
            else return 1;
    }

    // Everything has been all right
    return 0;
}

void GraphWidget::refreshPoints()
{

    QList<Node *> nodes;

    foreach (QGraphicsItem *item, scene()->items())

        if (Node *node = qgraphicsitem_cast<Node *>(item))
            if (node->helpNode == false) nodes << node;

    foreach (Node *node, nodes)
        node->calculateForces();

}
void GraphWidget::savePicture(const QString &fileName)
{

    QString pathImageWarped;

    if (fileName.length() == 0)
    {

        pathImageWarped.insert(0,path.mid(0,path.length()-4));
        pathImageWarped = pathImageWarped.insert(path.length()-4,"-warped.jpg");
    }

    else

        pathImageWarped.insert(0,fileName);


    //fprintf(stderr,"%s",pathImageWarped.toLatin1().data());
    imf2->save(pathImageWarped);

    // change the extension of the image to save the mesh file
    QString pathMesh;
    pathMesh = pathMesh.append(pathImageWarped.mid(0,pathImageWarped.length() - 3));
    pathMesh = pathMesh.append("dat");


    FILE* f=fopen(pathMesh.toLatin1().data(),"w");

    if (!f)
        {

        QMessageBox::warning(this, tr("Warp Faces"),
                             tr("Error saving mesh file: %1\nDo you have rights to do that?")
                             .arg(pathMesh));
            //fprintf(stderr,"Error saving the mesh!\n");
            exit(1);
        }

    // Creation of a list with the nodes to save

    QList<Node *> nodes;

    foreach (QGraphicsItem *item, scene()->items())
        if (Node *node = qgraphicsitem_cast<Node *>(item))

            // Ignore the help nodes of the right picture (only for help pourpose)
            if (node->helpNode == false) nodes << node;

    fprintf(f,"86\n");

    foreach (Node *node, nodes)
        fprintf(f,"%d %d %c\n",(int)node->posX(),(int)node->posY(),node->getColor());

    fclose(f);
}

void GraphWidget::downloadAddMesh(QList<qreal> *list)
{
    int i=0;
    QList<Node *> nodes;
    foreach (QGraphicsItem *item, scene()->items())
    {
        if (Node *node = qgraphicsitem_cast<Node *>(item))
        if (node->helpNode == false)
            nodes << node;
    }

    // the lenght of this list must be the same as 2*the number of nodes in the mesh
    if (list->size() == 68)
    {

        foreach (Node *node, nodes)
        {

            list->replace(i,list->value(i) + node->posX());
            list->replace(i+1,list->value(i+1) + node->posY());
            i = i + 2;
        }
    }
```

```
}

void GraphWidget::wheelEvent(QWheelEvent *event)
{
    scaleView(pow((double)2, -event->delta() / 240.0));
}

void GraphWidget::scaleView(qreal scaleFactor)
{
    qreal factor = matrix().scale(scaleFactor, scaleFactor).mapRect(QRectF(0, 0, 1, 1)).width();
    if (factor < 0.07 || factor > 100)
        return;

    scale(scaleFactor, scaleFactor);
}


void GraphWidget::increaseGroupDistance()
{
    int ii=0;char colorGroup='.';
    int  minpX = 1000, minpY = 1000;

    QList<Node *> nodes;
    foreach (QGraphicsItem *item, scene()->items())
    {
        if (Node *node = qgraphicsitem_cast<Node *>(item))
        if (node->helpNode == false) nodes << node;
    }
    foreach (Node *node, nodes)
        {
            if (ii == lastClicked) colorGroup = node->getColor();
            ii++;
        }

    foreach (Node *node, nodes)
        {
        if (node->getColor() == colorGroup)
            {
                if (node->posX() < minpX) minpX = node->posX();
                if (node->posY() < minpY) minpY = node->posY();
            }

        }

    foreach (Node *node, nodes)
        {
        if (node->getColor() == colorGroup)
            {
                // increasing the distance 20%
                node->setPos(node->posX() + 0.05 * ( node->posX() - minpX),
                             node->posY() + 0.05 * ( node->posY() - minpY));
            }
        }


}
void GraphWidget::turnGroupRight()
{

    int ii=0;char colorGroup='.';
    int  minpX = 1000, minpY = 1000, maxpX = 0, maxpY = 0;

    QList<Node *> nodes;
    foreach (QGraphicsItem *item, scene()->items())
    {
        if (Node *node = qgraphicsitem_cast<Node *>(item))
        if (node->helpNode == false) nodes << node;
    }
    foreach (Node *node, nodes)
        {
            if (ii == lastClicked) colorGroup = node->getColor();
            ii++;
        }

    foreach (Node *node, nodes)
        {
        if (node->getColor() == colorGroup)
            {
                if (node->posX() < minpX) minpX = node->posX();
                if (node->posY() < minpY) minpY = node->posY();
                if (node->posX() > maxpX) maxpX = node->posX();
                if (node->posY() > maxpY) maxpY = node->posY();
            }

        }

    if (centerGroupX == 0 and centerGroupY == 0)
        {
            centerGroupX =  (maxpX - minpX) / 2;
            centerGroupY =  (maxpY - minpY) / 2;
        }
    int hyp = 0 ;
    float theta = 0.0;
    foreach (Node *node, nodes)

    {

    if (node->getColor() == colorGroup)
        {
            // turn the group 15 degrees clockwise

            theta = atan((node->posY() - centerGroupY) / (node->posX() - centerGroupX)) + 15*M_PI/180;
            //printf("teta = %f\n", theta);

            hyp = sqrt((node->posX() - centerGroupX) * (node->posX() - centerGroupX) +
                       (node->posY() - centerGroupY) * (node->posY() - centerGroupY) );

            node->setPos(115 -centerGroupX + hyp * cos(theta) ,
                         115 -centerGroupY + hyp * sin(theta) );

        }
    }


}

void GraphWidget::turnGroupLeft()
{

    int ii=0;char colorGroup='.';
    int  minpX = 1000, minpY = 1000, maxpX = 0, maxpY = 0;

    QList<Node *> nodes;
    foreach (QGraphicsItem *item, scene()->items())
    {
        if (Node *node = qgraphicsitem_cast<Node *>(item))
        if (node->helpNode == false) nodes << node;
    }
    foreach (Node *node, nodes)
        {
            if (ii == lastClicked) colorGroup = node->getColor();
            ii++;
        }

    foreach (Node *node, nodes)
        {
        if (node->getColor() == colorGroup)
            {
                if (node->posX() < minpX) minpX = node->posX();
                if (node->posY() < minpY) minpY = node->posY();
                if (node->posX() > maxpX) maxpX = node->posX();
                if (node->posY() > maxpY) maxpY = node->posY();
            }

        }

    if (centerGroupX == 0 and centerGroupY == 0)
        {
            centerGroupX =  (maxpX - minpX) / 2;
            centerGroupY =  (maxpY - minpY) / 2;
        }

    int hyp = 0 ;
    float theta = 0.0;
    foreach (Node *node, nodes)

    {

    if (node->getColor() == colorGroup)
        {
            // turn the group 15 degrees clockwise

            theta = atan((node->posY() - centerGroupY) / (node->posX() - centerGroupX)) - 15*M_PI/180;
            //printf("teta = %f\n", theta);

            hyp = sqrt((node->posX() - centerGroupX) * (node->posX() - centerGroupX) +
                       (node->posY() - centerGroupY) * (node->posY() - centerGroupY) );

            node->setPos( -centerGroupX + hyp * cos(theta) ,
                          -centerGroupY + hyp * sin(theta) );

        }
    }




}

void GraphWidget::decreaseGroupDistance()
{

    int ii=0;char colorGroup='.';
    int minpX = 1000, minpY = 1000;

    QList<Node *> nodes;
    foreach (QGraphicsItem *item, scene()->items())
    {
        if (Node *node = qgraphicsitem_cast<Node *>(item))
        if (node->helpNode == false) nodes << node;
    }

    foreach (Node *node, nodes)
        {
            if (ii == lastClicked) colorGroup = node->getColor();
            ii++;
        }

    foreach (Node *node, nodes)
        {
        if (node->getColor() == colorGroup)
            {
                if (node->posX() < minpX) minpX = node->posX();
                if (node->posY() < minpY) minpY = node->posY();
            }

        }

    foreach (Node *node, nodes)
        {
```

```
        if (node->getColor() == colorGroup)
            {
                // increasing the distance 20%
                node->setPos(node->posX() - 0.05 * ( node->posX() - minpX),
                             node->posY() - 0.05 * ( node->posY() - minpY));
            }
        }


}

void GraphWidget::displaceNodeDown()
{
    // Creation of a list with the nodes of the Scene to interact with

    QList<Node *> nodes;
    foreach (QGraphicsItem *item, scene()->items())
        if (Node *node = qgraphicsitem_cast<Node *>(item))
            // Ignore the help nodes of the right picture (only for help pourpose)
            if (node->helpNode == false) nodes << node;

    int ii=0;

    foreach (Node *node, nodes)
        if (ii++ == lastClicked) node->setPos(node->posX(), node->posY() + 2);

}

void GraphWidget::displaceNodeUp()
{
    // Creation of a list with the nodes of the Scene to interact with

    QList<Node *> nodes;
    foreach (QGraphicsItem *item, scene()->items())
        if (Node *node = qgraphicsitem_cast<Node *>(item))
            // Ignore the help nodes of the right picture (only for help pourpose)
            if (node->helpNode == false) nodes << node;

    int ii=0;

    foreach (Node *node, nodes)
        if (ii++ == lastClicked) node->setPos(node->posX(), node->posY() - 2);

}

void GraphWidget::displaceNodeLeft()
{
    // Creation of a list with the nodes of the Scene to interact with

    QList<Node *> nodes;
    foreach (QGraphicsItem *item, scene()->items())
        if (Node *node = qgraphicsitem_cast<Node *>(item))
            // Ignore the help nodes of the right picture (only for help pourpose)
            if (node->helpNode == false) nodes << node;

    int ii=0;
    foreach (Node *node, nodes)
        if (ii++ == lastClicked) node->setPos(node->posX() - 2, node->posY());
}

void GraphWidget::displaceNodeRight()
{
    // Creation of a list with the nodes of the Scene to interact with

    QList<Node *> nodes;
    foreach (QGraphicsItem *item, scene()->items())
        if (Node *node = qgraphicsitem_cast<Node *>(item))
            // Ignore the help nodes of the right picture (only for help pourpose)
            if (node->helpNode == false) nodes << node;
    int ii=0;
    foreach (Node *node, nodes)
        if (ii++ == lastClicked) node->setPos(node->posX() + 2, node->posY());
}



void GraphWidget::displaceGroupLeft()
{
    // Creation of a list with the nodes of the Scene to interact with

    QList<Node *> nodes;
    foreach (QGraphicsItem *item, scene()->items())
        if (Node *node = qgraphicsitem_cast<Node *>(item))
            // Ignore the help nodes of the right picture (only for help pourpose)
            if (node->helpNode == false) nodes << node;


    int ii=0;
    char colorGroup='.';

    foreach (Node *node, nodes)
        if (ii++ == lastClicked) colorGroup = node->getColor();

    foreach (Node *node, nodes)
        if (node->getColor() == colorGroup) node->setPos(node->posX() - 2, node->posY());

}

void GraphWidget::displaceGroupRight()
{
    // Creation of a list with the nodes of the Scene to interact with

    QList<Node *> nodes;
    foreach (QGraphicsItem *item, scene()->items())
        if (Node *node = qgraphicsitem_cast<Node *>(item))
            // Ignore the help nodes of the right picture (only for help pourpose)
            if (node->helpNode == false) nodes << node;


    int ii=0;
    char colorGroup='.';

    foreach (Node *node, nodes)
        if (ii++ == lastClicked) colorGroup = node->getColor();

    foreach (Node *node, nodes)
        if (node->getColor() == colorGroup) node->setPos(node->posX() + 2, node->posY());

}


void GraphWidget::displaceGroupDown()
{
    // Creation of a list with the nodes of the Scene to interact with

    QList<Node *> nodes;
    foreach (QGraphicsItem *item, scene()->items())
        if (Node *node = qgraphicsitem_cast<Node *>(item))
            // Ignore the help nodes of the right picture (only for help pourpose)
            if (node->helpNode == false) nodes << node;

    int ii=0;
    char colorGroup='.';

    foreach (Node *node, nodes)
        if (ii++ == lastClicked) colorGroup = node->getColor();

    foreach (Node *node, nodes)
        if (node->getColor() == colorGroup) node->setPos(node->posX(), node->posY() + 2);
```

```
}

void GraphWidget::displaceGroupUp()
{
    // Creation of a list with the nodes of the Scene to interact with

    QList<Node *> nodes;
    foreach (QGraphicsItem *item, scene()->items())
        if (Node *node = qgraphicsitem_cast<Node *>(item))
            // Ignore the help nodes of the right picture (only for help pourpose)
            if (node->helpNode == false) nodes << node;


    int ii=0;
    char colorGroup='.';

    foreach (Node *node, nodes)
        if (ii++ == lastClicked) colorGroup = node->getColor();

    foreach (Node *node, nodes)
        if (node->getColor() == colorGroup) node->setPos(node->posX(), node->posY() - 2);

}

void GraphWidget::displaceAllLeft()
{
    // Creation of a list with the nodes of the Scene to interact with

    QList<Node *> nodes;
    foreach (QGraphicsItem *item, scene()->items())
        if (Node *node = qgraphicsitem_cast<Node *>(item))
            // Ignore the help nodes of the right picture (only for help pourpose)
            if (node->helpNode == false) nodes << node;


    foreach (Node *node, nodes)
        if (!(node->getColor() == 't'))
            node->setPos(node->posX() - 2, node->posY());

}

void GraphWidget::displaceAllRight()
{
    // Creation of a list with the nodes of the Scene to interact with

    QList<Node *> nodes;
    foreach (QGraphicsItem *item, scene()->items())
        if (Node *node = qgraphicsitem_cast<Node *>(item))
            // Ignore the help nodes of the right picture (only for help pourpose)
            if (node->helpNode == false) nodes << node;

    foreach (Node *node, nodes)
        if (!(node->getColor() == 't'))
            node->setPos(node->posX() + 2, node->posY());

}

void GraphWidget::displaceAllUp()
{

    // Creation of a list with the nodes of the Scene to interact with

    QList<Node *> nodes;
    foreach (QGraphicsItem *item, scene()->items())
        if (Node *node = qgraphicsitem_cast<Node *>(item))
            // Ignore the help nodes of the right picture (only for help pourpose)
            if (node->helpNode == false) nodes << node;

    foreach (Node *node, nodes)
```

```
        if (!(node->getColor() == 't'))
            node->setPos(node->posX(), node->posY() - 2);

}

void GraphWidget::displaceAllDown()
{
    // Creation of a list with the nodes of the Scene to interact with

    QList<Node *> nodes;
    foreach (QGraphicsItem *item, scene()->items())
        if (Node *node = qgraphicsitem_cast<Node *>(item))
            // Ignore the help nodes of the right picture (only for help pourpose)
            if (node->helpNode == false) nodes << node;

    foreach (Node *node, nodes)
        if (!(node->getColor() == 't'))
            node->setPos(node->posX(), node->posY() + 2);

}
int GraphWidget::updatePicture(bool showInWindow)
{

    //Pre selection of the size of the head 180 x 180
    int mTriangles[ imf1->width()][imf1->height()];
    bool flag = false;

    int RedInterpolated, GreenInterpolated, BlueInterpolated;
    // For the pixel interpolation:
    double dyUp,dyDown,dxLeft,dxRight;

    //Creation of a QProgressDialog to let the user wait patiently :)
    QProgressDialog * pbd = new QProgressDialog(QString("Warping picture...please wait!"),
                                    QString("&Cancel operation"),0,100, this);
    pbd->setAutoClose(true);


    for (int j = 0; j < imf1->height();j++){
        for (int i = 0; i < imf1->width();i++){
            // For each triangle in the image...
            for (int k = 0; k < nTri; k ++)
                {

                    inside = insideTriangle(aNodes2[ aTri[k][0]][0], aNodes2[ aTri[k][0]][1],
                                          aNodes2[ aTri[k][1]][0], aNodes2[ aTri[k][1]][1],
                                          aNodes2[ aTri[k][2]][0], aNodes2[ aTri[k][2]][1],i,j);
                    if (inside == true)
                    {
                        //printf("mTriangles[i=%d][j=%d] = %d\n",i,j,k);
                        mTriangles[i][j] = k;
                        flag = true;
                    }
                }
            if (flag == false)
            {
                QMessageBox::warning(this, tr("Fatal error warping the mesh"),
                          tr("It seems that there are some points of the mesh that cannot be reached.\n Exi
ting..."));
                printf("Point out of matrix ?? point X(%d,%d)\n",i, j);
                return(-1);

            }
            flag = false;
        }
    }




    // Now, creation of the matrix and compute the inverse...
```

```
for (int j = 0; j < imf1->height()-1;j++){
for (int i = 0; i < imf1->width()-1; i++){


    //Point A of the triangle in which the point (i,j) is located
    A[0][0] = aNodes2 [aTri[ mTriangles[i][j] ][0]][0];
    A[1][0] = aNodes2 [aTri[ mTriangles[i][j] ][0]][1];
    A[2][0] = 1;

    //Point B of the triangle in which the point (i,j) is located
    A[0][1] = aNodes2 [aTri[ mTriangles[i][j] ][1]][0];
    A[1][1] = aNodes2 [aTri[ mTriangles[i][j] ][1]][1];
    A[2][1] = 1;

    //Point C of the triangle in which the point (i,j) is located
    A[0][2] = aNodes2 [aTri[ mTriangles[i][j] ][2]][0];
    A[1][2] = aNodes2 [aTri[ mTriangles[i][j] ][2]][1];
    A[2][2] = 1;


//Inverse....

detInv = (  A[0][0] * (A[1][1] * A[2][2] - A[1][2] * A[2][1]) -
        A[0][1] * (A[1][0] * A[2][2] - A[1][2] * A[2][0]) +
        A[0][2] * (A[1][0] * A[2][1] - A[1][1] * A[2][0]));
    if( detInv == 0.0 ) {printf("Determinant null! %d %d %f\n",i,j, A[1][1]);break;}

    detInv = (double)(1.0 / detInv);
    in[0][0] =  detInv * (A[1][1] * A[2][2] - A[1][2] * A[2][1]);
    in[0][1] = -detInv * (A[0][1] * A[2][2] - A[0][2] * A[2][1]);
    in[0][2] =  detInv * (A[0][1] * A[1][2] - A[0][2] * A[1][1]);

    in[1][0] = -detInv * (A[1][0] * A[2][2] - A[1][2] * A[2][0]);
    in[1][1] =  detInv * (A[0][0] * A[2][2] - A[0][2] * A[2][0]);
    in[1][2] = -detInv * (A[0][0] * A[1][2] - A[0][2] * A[1][0]);

    in[2][0] =  detInv * (A[1][0] * A[2][1] - A[1][1] * A[2][0]);
    in[2][1] = -detInv * (A[0][0] * A[2][1] - A[0][1] * A[2][0]);
    in[2][2] =  detInv * (A[0][0] * A[1][1] - A[0][1] * A[1][0]);


//Then, once inverted, multiply by the values of the position of X(i,j)

    alpha = in[0][0]*i + in[0][1]*j + in[0][2];
    beta =  in[1][0]*i + in[1][1]*j + in[1][2];
    gamma = in[2][0]*i + in[2][1]*j + in[2][2];

//Now

    //Point Ap of the triangle (original triangle)
    Apx = aNodes [aTri[ mTriangles[i][j] ][0]][0];
    Apy = aNodes [aTri[ mTriangles[i][j] ][0]][1];

    //Point Bp of the triangle (original triangle)
    Bpx = aNodes [aTri[ mTriangles[i][j] ][1]][0];
    Bpy = aNodes [aTri[ mTriangles[i][j] ][1]][1];

    //Point Cp of the triangle (original triangle)
    Cpx = aNodes [aTri[ mTriangles[i][j] ][2]][0];
    Cpy = aNodes [aTri[ mTriangles[i][j] ][2]][1];


    Xp = alpha * Apx + beta * Bpx + gamma * Cpx;
    Yp = alpha * Apy + beta * Bpy + gamma * Cpy;


    // Interpolation of the pixel


    // Nearest Neighbour Method  .... or ...
```

```
        // valueOrig = imf1->pixel(Xp,Yp);

        //Bicubic interpolation:

        dxLeft =  Xp - int(Xp);
        dxRight = 1 -dxLeft;
        dyUp = Yp - int(Yp);
        dyDown = 1 - dyUp;


        // Get the interpolated colours:
        RedInterpolated = qRed(imf1->pixel((int)Xp  ,(int)Yp  )) * (dxRight * dyDown) +
                          qRed(imf1->pixel((int)Xp  ,(int)Yp+1)) * (dxRight * dyUp  ) +
                          qRed(imf1->pixel((int)Xp+1,(int)Yp  )) * (dxLeft * dyDown ) +
                          qRed(imf1->pixel((int)Xp+1,(int)Yp+1)) * (dxLeft * dyUp   );

        GreenInterpolated = qGreen(imf1->pixel((int)Xp  ,(int)Yp  )) * (dxRight * dyDown) +
                            qGreen(imf1->pixel((int)Xp  ,(int)Yp+1)) * (dxRight * dyUp  ) +
                            qGreen(imf1->pixel((int)Xp+1,(int)Yp  )) * (dxLeft * dyDown ) +
                            qGreen(imf1->pixel((int)Xp+1,(int)Yp+1)) * (dxLeft * dyUp   );

        BlueInterpolated = qBlue(imf1->pixel((int)Xp  ,(int)Yp  )) * (dxRight * dyDown) +
                           qBlue(imf1->pixel((int)Xp  ,(int)Yp+1)) * (dxRight * dyUp  ) +
                           qBlue(imf1->pixel((int)Xp+1,(int)Yp  )) * (dxLeft * dyDown ) +
                           qBlue(imf1->pixel((int)Xp+1,(int)Yp+1)) * (dxLeft * dyUp   );

        // Create a color with the values obtained and fill the pixel
        QColor color(RedInterpolated, GreenInterpolated, BlueInterpolated);
        imf2->setPixel(i,j,color.rgb());
        refreshPoints();
        if (pbd->wasCanceled() == true) return 1;
        pbd->setValue((int)(100*j/imf1->height() +1));
    }
    }



    if (showInWindow == true) showWarpedWindow(imf2);
    //printf("\n> image warped successfully!\n");
    return (0);
}



void GraphWidget::nodeClicked(int nodeNumber)
{

    // Positions of the nodes before removing them
    qreal newHelpX, newHelpY, oldHelpX = 0, oldHelpY = 0;

    // Store the position of the new red node
    newHelpX = hNodes[nodeNumber][0];
    newHelpY = hNodes[nodeNumber][1];

    // Check if it is the first time or there is a previous clicked node
    // If there is a previous red one remove it
    if (lastClicked != -1)
    {
        oldHelpX = hNodes[lastClicked][0];
        oldHelpY = hNodes[lastClicked][1];
        superScene->removeItem(hnode[lastClicked]);
    }
    // Remove the new one to change its color
    superScene->removeItem(hnode[nodeNumber]);

    // Create a new node with the normal color;
    if (lastClicked != -1)
    {
        hnode[lastClicked]= new Node(this,lastClicked,'m',true);
        superScene->addItem(hnode[lastClicked]);
        // Displacement of the point
        hnode[lastClicked]->setPos(oldHelpX,oldHelpY);
    }

    // Create a new node with the red color;
    hnode[nodeNumber]= new Node(this,nodeNumber,'r',true);
    superScene->addItem(hnode[nodeNumber]);
    // Displacement of the point
    hnode[nodeNumber]->setPos(newHelpX,newHelpY);

    lastClicked = nodeNumber;

}


int GraphWidget::detectElements(bool stretch)
{

    // Sub-images with the location of the elements
    IplImage *subImageFace, *subImageEyeR, *subImageEyeL, *subImageNose, *subImageMouth, *subImageEyes;
    // Detectors
    Detector  *dFace, *dEyeR, *dEyeL, *dMouth, *dNose, *dEyes;

    // General variables for elements recognition
    int px = 0,py = 0,imHeight = 0,imWidth = 0;
    int centerX, centerY;
    int h = 0;
    int nodei = 0; // for list moving

    // Variables to obtain the boundary conditions

    // Face Coordinates
        int faceX, faceY, faceW, faceH;

    // Both eyes Coordinates
        int eyesX, eyesY, eyesW, eyesH;

    // Left eye Coordinates
        int lEyeX, lEyeY, lEyeW, lEyeH;
        int lEyeMediumX, lEyeMediumY;

    // Right eye Coordinates
        int rEyeX, rEyeY, rEyeW, rEyeH;
        int rEyeMediumX, rEyeMediumY;

    // Mouth Coordinates
        int mouthX, mouthY, mouthW, mouthH;

    // Nose Coordinates
        int noseX, noseY, noseW, noseH;


    // Creation of a list with the nodes to be moved (not help nodes)

    QList<Node *> nodes;
    foreach (QGraphicsItem *item, scene()->items()) {
        if (Node *node = qgraphicsitem_cast<Node *>(item))
        if (node->helpNode == false) nodes << node;
    }


    // ##################################################################################
    // -------------------- First: Face recognition  ------------------------------------
    // ##################################################################################

    // Creation of the subImage that will contain the face

    subImageFace = cvCreateImage(cvSize(imf1->width(), imf1->height()), IPL_DEPTH_8U, 3);
    for (int i = 0; i < subImageFace->height;i++)
    {
        for (int j = 0; j < subImageFace->width*3; j+=3)
        {

            ((char*)subImageFace->imageData)[i*imf1->width()*3 + j + 0] = qBlue(imf1->pixel(h,i));    // b
lue channel
            ((char*)subImageFace->imageData)[i*imf1->width()*3 + j + 1] = qGreen(imf1->pixel(h,i));  // gr
een channel
            ((char*)subImageFace->imageData)[i*imf1->width()*3 + j + 2] = qRed(imf1->pixel(h,i));   // re
d channel
            h++;
        }h=0;
    }

    //cvSaveImage("face.jpg",subImageFace);
    //Creation of the detector object and obtention of the coordinates
    dFace = new Detector(subImageFace);
    dFace->DetectAndDraw('F',&px,&py,&imHeight,&imWidth);

    // Display a rectangle with the face location
    superScene->addRect((qreal)px,(qreal)py,(qreal)imWidth,(qreal)imHeight,*(new QPen(*(new QColor(0,0,
0)))));
    // Debugging console message
    fprintf(stderr,"Coordn face %d %d hei %d wid %d\n",px,py,imHeight,imWidth);

    // Fill the face boundary variables. Throught the face boundaries the rest of the elements will be pro
vided
    faceX = px; faceY = py; faceW = imWidth; faceH = imHeight;

    // Check there is a face recogniced to continue
    if (px == 0 && py == 0 && imWidth == 0 && imHeight == 0) return 1;

    // Move the nodes where we think they should be...


    // ####################################################################################
    // -------- Second: Both eyes recognition upper half face -------------------
    // ####################################################################################


    subImageEyes = cvCreateImage(cvSize(imf1->width(), imf1->height()), IPL_DEPTH_8U, 3);

    // Clean the image
    for (int i = 0; i < subImageEyes->width*subImageEyes->height*3; i++) // over w*h*3 (channels)
            ((char*)subImageEyes->imageData)[i] = 255;

    // Copy the information between both image formats. Only the face region is needed to locate the eye s
quares
    h=faceX;
    for (int i = faceY ; i < faceY + faceH*3/4;i++)
    {
        for (int j = 3* faceX; j <  (faceX + faceW)*3; j+=3)
        {

            ((char*)subImageEyes->imageData)[i*imf1->width()*3 + j + 0] = qBlue(imf1->pixel(h,i));    // b
lue channel
            ((char*)subImageEyes->imageData)[i*imf1->width()*3 + j + 1] = qGreen(imf1->pixel(h,i));  // gr
een channel
            ((char*)subImageEyes->imageData)[i*imf1->width()*3 + j + 2] = qRed(imf1->pixel(h,i));   // re
d channel
            h++;
        }h=faceX;
    }


    // Save the image (debugging)
    //cvSaveImage("eyes.jpg",subImageEyes);

    //Creation of the detector object and obtention of the coordinates
    dEyes = new Detector(subImageEyes);
    dEyes->DetectAndDraw('E',&px,&py,&imHeight,&imWidth);
    superScene->addRect((qreal)px,(qreal)py,(qreal)imWidth,(qreal)imHeight,  *(new QPen( * (new QColor(25
5,0,255)))));
    fprintf(stderr,"Coordn eyes %d %d hei %d wid %d\n",px,py,imHeight,imWidth);

    // Fill the eyes boundary variables. Throught the face boundaries the eyes will be provided
    eyesX = px; eyesY = py; eyesW = imWidth; eyesH = imHeight;

    // Check there are eyes to continue
    if (px == 0 && py == 0 && imWidth == 0 && imHeight == 0) return 1;




    // #####################################################################################
    // -------- Third: Left eye recognition inside left half eyes region  ------------------
    // #####################################################################################


    subImageEyeL = cvCreateImage(cvSize(imf1->width(), imf1->height()), IPL_DEPTH_8U, 3);

    // Clean the image
    for (int i = 0; i < subImageEyeL->width*subImageEyeL->height*3; i++) // over w*h*3 (channels)
            ((char*)subImageEyeL->imageData)[i] = 255;

    // Copy the information between both image formats. Only the face region is needed to locate the eye s
quares
    h=eyesX;
    for (int i = eyesY ; i < eyesY + eyesH;i++)
    {
        for (int j = 3* eyesX; j <  (eyesX + eyesW/2)*3; j+=3)
        {

            ((char*)subImageEyeL->imageData)[i*imf1->width()*3 + j + 0] = qBlue(imf1->pixel(h,i));    // b
lue channel
            ((char*)subImageEyeL->imageData)[i*imf1->width()*3 + j + 1] = qGreen(imf1->pixel(h,i));  // gr
een channel
            ((char*)subImageEyeL->imageData)[i*imf1->width()*3 + j + 2] = qRed(imf1->pixel(h,i));   // re
d channel
            h++;
        }h=eyesX;
    }


    // Save the image (debugging)
    //cvSaveImage("left.jpg",subImageEyeL);

    //Creation of the detector object and obtention of the coordinates
    dEyeL = new Detector(subImageEyeL);
    dEyeL->DetectAndDraw('L',&px,&py,&imHeight,&imWidth);
    superScene->addRect((qreal)px,(qreal)py,(qreal)imWidth,(qreal)imHeight,  *(new QPen( * (new QColor(25
5,0,0)))));
    fprintf(stderr,"Coordn eye Left %d %d hei %d wid %d\n",px,py,imHeight,imWidth);

    // Fill the face boundary variables.
    lEyeX = px; lEyeY = py; lEyeW = imWidth; lEyeH = imHeight;
    lEyeMediumX = 3*lEyeX + 0.5 * lEyeW; lEyeMediumY = lEyeY + 0.5 * lEyeH;
    // Check there is a left eye to continue
    if (px == 0 && py == 0 && imWidth == 0 && imHeight == 0) return 1;

    nodei = 0;

    // Select the allocation mode: stretch or not stretch
    if (stretch == false)
    foreach (Node *node, nodes)
    {
        if (nodei == 44) node->setPos(lEyeX + 0.05 * imWidth, lEyeY + 0.50 * imHeight);
        if (nodei == 45) node->setPos(lEyeX + 0.95 * imWidth, lEyeY + 0.50 * imHeight);
        if (nodei == 43) node->setPos(lEyeX + 0.50 * imWidth, lEyeY + 0.05 * imHeight);
        if (nodei == 46) node->setPos(lEyeX + 0.50 * imWidth, lEyeY + 0.95 * imHeight);
        if (nodei == 40) node->setPos(lEyeX + 0.25 * imWidth, lEyeY + 0.60 * imHeight);
        if (nodei == 41) node->setPos(lEyeX + 0.75 * imWidth, lEyeY + 0.60 * imHeight);
        if (nodei == 38) node->setPos(lEyeX + 0.25 * imWidth, lEyeY + 0.40 * imHeight);
        if (nodei == 39) node->setPos(lEyeX + 0.75 * imWidth, lEyeY + 0.40 * imHeight);
        if (nodei == 37) node->setPos(lEyeX + 0.50 * imWidth, lEyeY + 0.45 * imHeight);
        if (nodei == 42) node->setPos(lEyeX + 0.15 * imWidth, lEyeY + 0.10 * imHeigh
t);
        if (nodei == 47) node->setPos(lEyeX + 0.90 * imWidth, lEyeY + 0.70 * imHeigh
t);
        nodei++;
    }
    else
    {
    centerX = px + imWidth / 2;centerY = py + imHeight/2;
        foreach (Node *node, nodes)
        {
            if (nodei == 44) node->setPos(centerX - 37, centerY -  1 );
            if (nodei == 45) node->setPos(centerX + 37, centerY -  1 );
            if (nodei == 43) node->setPos(centerX +  0, centerY - 22 );
            if (nodei == 46) node->setPos(centerX +  0, centerY + 22 );
            if (nodei == 40) node->setPos(centerX - 12, centerY +  4 );
            if (nodei == 41) node->setPos(centerX + 18, centerY +  2 );
            if (nodei == 38) node->setPos(centerX - 11, centerY - 11 );
            if (nodei == 39) node->setPos(centerX + 18, centerY - 11 );
            if (nodei == 37) node->setPos(centerX +  3, centerY -  8 );
            if (nodei == 42) node->setPos(centerX - 22, centerY - 14 );
            if (nodei == 47) node->setPos(centerX + 31, centerY + 11 );
            nodei++;
        }
    }


    // #####################################################################################
    // -------- Fourth: Right eye recognition inside left upper quarter  -------------------
    // #####################################################################################


    subImageEyeR = cvCreateImage(cvSize(imf1->width(), imf1->height()), IPL_DEPTH_8U, 3);

    // Clean the image
    for (int i = 0; i < subImageEyeR->width*subImageEyeR->height*3; i++) // over w*h*3 (channels)
        ((char*)subImageEyeR->imageData)[i] = 255;

    // Copy the information between both image formats. Only the face region is needed to locate the eye s
quares
    h = eyesX + eyesW/2;
    for (int i = eyesY ; i < eyesY + eyesH;i++)
    {
        for (int j = 3*(eyesX + eyesW/2); j <  3*(eyesX + eyesW); j+=3)
        {

            ((char*)subImageEyeR->imageData)[i*imf1->width()*3 + j  + 0] = qBlue(imf1->pixel(h,i));    //
blue channel
            ((char*)subImageEyeR->imageData)[i*imf1->width()*3 + j  + 1] = qGreen(imf1->pixel(h,i));  // g
reen channel
            ((char*)subImageEyeR->imageData)[i*imf1->width()*3 + j  + 2] = qRed(imf1->pixel(h,i));   // re
d channel
            h++;
        }h=eyesX + eyesW/2;
    }
   // cvSaveImage("right.jpg",subImageEyeR);
    dEyeR = new Detector(subImageEyeR);
    dEyeR->DetectAndDraw('R',&px,&py,&imHeight,&imWidth);
    superScene->addRect((qreal)px,(qreal)py,(qreal)imWidth,(qreal)imHeight,*(new QPen(*(new QColor(0,0,25
5)))));
    fprintf(stderr,"Coordn eye Right %d %d wid %d hei %d\n",px,py,imWidth,imHeight);

    // Fill the face boundary variables.
    rEyeX = px; rEyeY = py; rEyeW = imWidth; rEyeH = imHeight;
    rEyeMediumX = 3*rEyeX + 0.5 * rEyeW; rEyeMediumY = rEyeY + 0.5 * rEyeH;
    // Check there is a right eye to continue
    if (px == 0 && py == 0 && imWidth == 0 && imHeight == 0) return 1;

    // Move the nodes where we think they should be... important, an offset of 0.05*size is added
   nodei = 0;

    // Select the allocation mode: stretch or not stretch
    if (stretch == false)
    foreach (Node *node, nodes)
    {

        if (nodei == 56) node->setPos(px + 0.05 * imWidth, py + 0.50 * imHeight);
        if (nodei == 55) node->setPos(px + 0.95 * imWidth, py + 0.50 * imHeight);
        if (nodei == 54) node->setPos(px + 0.50 * imWidth, py + 0.05 * imHeight);
        if (nodei == 57) node->setPos(px + 0.50 * imWidth, py + 0.95 * imHeight);
        if (nodei == 52) node->setPos(px + 0.25 * imWidth, py + 0.60 * imHeight);
        if (nodei == 51) node->setPos(px + 0.75 * imWidth, py + 0.60 * imHeight);
        if (nodei == 50) node->setPos(px + 0.25 * imWidth, py + 0.40 * imHeight);
        if (nodei == 49) node->setPos(px + 0.75 * imWidth, py + 0.40 * imHeight);
        if (nodei == 48) node->setPos(px + 0.50 * imWidth, py + 0.45 * imHeight);
        if (nodei == 53) node->setPos(px + 0.85 * imWidth, py + 0.10 * imHeight);
        if (nodei == 58) node->setPos(px + 0.10 * imWidth, py + 0.70 * imHeight);


        // Position of the node between both eyes
        //    x = average between X coordinate for both eyes
        //    y = average between Y coordinate for both eyes

        //if (nodei == 36) node->setPos((rEyeMediumX + lEyeMediumX)/2, (rEyeMediumY + lEyeMediumY)/2);
        nodei++;
    }
    else
    {
    centerX = px + imWidth / 2;centerY = py + imHeight/2;
        foreach (Node *node, nodes)
        {
            if (nodei == 56) node->setPos(centerX - 37, centerY -  1 );
            if (nodei == 55) node->setPos(centerX + 37, centerY -  1 );
            if (nodei == 54) node->setPos(centerX +  0, centerY - 22 );
            if (nodei == 57) node->setPos(centerX +  0, centerY + 22 );
            if (nodei == 52) node->setPos(centerX - 18, centerY +  4 );
            if (nodei == 51) node->setPos(centerX + 12, centerY +  2 );
            if (nodei == 50) node->setPos(centerX - 12, centerY - 11 );
            if (nodei == 49) node->setPos(centerX + 11, centerY - 11 );
            if (nodei == 48) node->setPos(centerX -  3, centerY -  8 );
            if (nodei == 53) node->setPos(centerX + 22, centerY - 14 );
            if (nodei == 58) node->setPos(centerX - 31, centerY + 11 );
        if (nodei == 36) node->setPos((rEyeMediumX + lEyeMediumX)/2, (rEyeMediumY + lEyeMediumY)/
2);
            nodei++;
        }
    }
    // ####################################################################################
    // -------- Fifth: Mouth recognition in the center side  -------------------
    // ####################################################################################


    subImageMouth = cvCreateImage(cvSize(imf1->width(), imf1->height()), IPL_DEPTH_8U, 3);

    // Clean the image
    for (int i = 0; i < subImageMouth->width*subImageMouth->height*3; i++) // over w*h*3 (channels)
        ((char*)subImageMouth->imageData)[i] = 255;

    // Copy the information between both image formats. Only the face region is needed to locate the eye s
quares
    h=faceX;
    for (int i = faceY + faceH*1/2 ; i < faceY + faceH;i++)
    {
        for (int j = 3*faceX; j <  (faceX + faceW)*3; j+=3)
        {
            ((char*)subImageMouth->imageData)[i*imf1->width()*3 + j  + 0] = qBlue(imf1->pixel(h,
i));    // blue channel
            ((char*)subImageMouth->imageData)[i*imf1->width()*3 + j  + 1] = qGreen(imf1->pixel(h,i));  //
green channel
            ((char*)subImageMouth->imageData)[i*imf1->width()*3 + j  + 2] = qRed(imf1->pixel(h,i));   // r
ed channel
            h++;
        }h=faceX;
    }
    cvSaveImage("mouth.jpg",subImageMouth);
    dMouth = new Detector(subImageMouth);
    dMouth->DetectAndDraw('M',&px,&py,&imHeight,&imWidth);
    superScene->addRect((qreal)px,(qreal)py,(qreal)imWidth,(qreal)imHeight,  *(new QPen( * (new QColor(0,2
55,255)))));
    fprintf(stderr,"Coordn mouth  %d %d wid %d hei %d\n",px,py,imWidth,imHeight);

    // Fill the face boundary variables.
    mouthX = px; mouthY = py; mouthW = imWidth; mouthH = imHeight;

    // Check there is a mouth
    if (px == 0 && py == 0 && imWidth == 0 && imHeight == 0) return 1;

    // Move the nodes where we think they should be... important, an offset is added
   nodei = 0;

    if (stretch == false)
    foreach (Node *node, nodes)
    {
        if (nodei == 26) node->setPos(px + 0.05 * imWidth, py + 0.50 * imHeight);
        if (nodei == 27) node->setPos(px + 1.00 * imWidth, py + 0.50 * imHeight);
        if (nodei == 23) node->setPos(px + 0.50 * imWidth, py + 0.10 * imHeight);
        if (nodei == 14) node->setPos(px + 0.50 * imWidth, py + 1.00 * imHeight);
        if (nodei == 24) node->setPos(px + 0.35 * imWidth, py + 0.05 * imHeight);
        if (nodei == 25) node->setPos(px + 0.65 * imWidth, py + 0.05 * imHeight);
        if (nodei == 18) node->setPos(px + 0.20 * imWidth, py + 0.50 * imHeight);
        if (nodei == 19) node->setPos(px + 0.80 * imWidth, py + 0.50 * imHeight);
        if (nodei == 21) node->setPos(px + 0.35 * imWidth, py + 0.20 * imHeight);
        if (nodei == 20) node->setPos(px + 0.50 * imWidth, py + 0.20 * imHeight);
        if (nodei == 22) node->setPos(px + 0.65 * imWidth, py + 0.20 * imHeight);
        if (nodei == 17) node->setPos(px + 0.50 * imWidth, py + 0.50 * imHeight);
        if (nodei == 20) node->setPos(px + 0.50 * imWidth, py + 0.25 * imHeight);
        if (nodei == 15) node->setPos(px + 0.25 * imWidth, py + 1.00 * imHeight);
        if (nodei == 16) node->setPos(px + 0.75 * imWidth, py + 1.00 * imHeight);
        nodei++;
    }
    else
    {
    centerX = px + imWidth / 2;centerY = py + imHeight/2;
        foreach (Node *node, nodes)
        {
            if (nodei == 26) node->setPos(centerX - 52, centerY -  4);
            if (nodei == 27) node->setPos(centerX + 52, centerY -  3);
            if (nodei == 23) node->setPos(centerX +  0, centerY - 16);
            if (nodei == 14) node->setPos(centerX +  0, centerY + 16);
            if (nodei == 24) node->setPos(centerX - 17, centerY - 18);
            if (nodei == 25) node->setPos(centerX + 17, centerY - 18);
            if (nodei == 18) node->setPos(centerX - 32, centerY -  3);
            if (nodei == 19) node->setPos(centerX + 32, centerY -  3);
            if (nodei == 21) node->setPos(centerX - 17, centerY -  9);
            if (nodei == 20) node->setPos(centerX +  0, centerY -  7);
            if (nodei == 22) node->setPos(centerX + 17, centerY -  9);
            if (nodei == 17) node->setPos(centerX +  0, centerY +  0);
            if (nodei == 20) node->setPos(centerX +  0, centerY -  7);
            if (nodei == 15) node->setPos(centerX - 29, centerY + 16);
            if (nodei == 16) node->setPos(centerX + 29, centerY + 16);
            nodei++;
        }
    }
    // ####################################################################################
    // -------- Sixth: Nose recognition inside the center side  -------------------
    // ###################################################################################

return 0;
    subImageNose = cvCreateImage(cvSize(imf1->width(), imf1->height()), IPL_DEPTH_8U, 3);

    // Clean the image
    for (int i = 0; i < subImageNose->width*subImageNose->height*3; i++) // over w*h*3 (channels)
        ((char*)subImageNose->imageData)[i] = 255;

    // Copy the information between both image formats. Only the face region is needed to locate the eye s
quares
    h=faceX;
    if (mouthY == 0 ) mouthY = faceY + faceH;
    for (int i = eyesY + eyesH ; i < mouthY;i++)
    {
        for (int j = 3*faceX; j <  (faceX + faceW)*3; j+=3)
        {

            ((char*)subImageNose->imageData)[i*imf1->width()*3 + j  + 0] = qBlue(imf1->pixel(h,i));    //
blue channel
            ((char*)subImageNose->imageData)[i*imf1->width()*3 + j  + 1] = qGreen(imf1->pixel(h,i));  // g
reen channel
            ((char*)subImageNose->imageData)[i*imf1->width()*3 + j  + 2] = qRed(imf1->pixel(h,i));   // re
d channel
            h++;
        }h=faceX;
    }
    //cvSaveImage("nose.jpg",subImageNose);
    dNose = new Detector(subImageNose);
    dNose->DetectAndDraw('N',&px,&py,&imHeight,&imWidth);
    superScene->addRect((qreal)px,(qreal)py,(qreal)imWidth,(qreal)imHeight,  *(new QPen( * (new QColor(12
7,0,127)))));
    fprintf(stderr,"Coordn nose  %d %d wid %d hei %d\n",px,py,imWidth,imHeight);

    // Fill the face boundary variables.
    noseX = px; noseY = py; noseW = imWidth; noseH = imHeight;

    // Check there is a nose
    if (px == 0 && py == 0 && imWidth == 0 && imHeight == 0) return 1;


    // Move the nodes where we think they should be... important, an offset of 0.10*size is added
   nodei = 0;
    if ((px != 0) and (py != 0))
    {
        if (stretch == false)
        {
            foreach (Node *node, nodes)
            {

                if (nodei == 31) node->setPos(px + 0.00 * imWidth, py + 0.50 * imHeight);
                if (nodei == 32) node->setPos(px + 1.00 * imWidth, py + 0.50 * imHeight);
                if (nodei == 34) node->setPos(px + 0.20 * imWidth, py + 0.05 * imHeight);
                if (nodei == 35) node->setPos(px + 0.80 * imWidth, py + 0.05 * imHeight);
                if (nodei == 33) node->setPos(px + 0.50 * imWidth, py + 0.20 * imHeight);
                if (nodei == 29) node->setPos(px + 0.25 * imWidth, py + 0.95 * imHeight);
                if (nodei == 30) node->setPos(px + 0.75 * imWidth, py + 0.95 * imHeight);
                if (nodei == 28) node->setPos(px + 0.50 * imWidth, py + 1.00 * imHeight);

                node->calculateForces();
                //if (nodei == 6) node->setPos((faceX+faceW),midYeyeR );
                nodei++;

            }
        }
        else
        {
        centerX = px + imWidth / 2;centerY = py + imHeight/2;
            foreach (Node *node, nodes)
            {
                if (nodei == 31) node->setPos(centerX - 42, centerY +  2);
                if (nodei == 32) node->setPos(centerX + 42, centerY +  2);
                if (nodei == 34) node->setPos(centerX - 27, centerY - 18);
                if (nodei == 35) node->setPos(centerX + 27, centerY - 18);
                if (nodei == 33) node->setPos(centerX +  0, centerY -  6);
                if (nodei == 29) node->setPos(centerX - 23, centerY + 11);
                if (nodei == 30) node->setPos(centerX + 23, centerY + 11);
                if (nodei == 28) node->setPos(centerX +  0, centerY + 18);
                if (nodei == 28) node->setPos(centerX +  0, centerY + 18);
                if (nodei == 36) node->setPos((rEyeMediumX + lEyeMediumX)/2, (rEyeMediumY + lEyeMediumY)/
2);
                nodei++;
            }
        }
    }

return 0;


}


long GraphWidget::lineLength(int A1, int B1, int A2, int B2)
{
    long X1 = A2 - A1;
    long X2 = B2 - B1;
    return sqrt( X2 * X2 + X1 * X1);
}

void GraphWidget::printEdges()
{

    showlines = true;
    refreshPoints();
    removeEdges();
    for (int i  = 0 ; i < 3; i++)
        for (int k = 0; k < nTri; k ++)
        {
            superScene->addLine(aNodes[ aTri[k][0]][0], aNodes[ aTri[k][0]][1],
                aNodes[ aTri[k][1]][0], aNodes[ aTri[k][1]][1],QPen( QColor(0,255,0) ) );
            superScene->addLine(aNodes[ aTri[k][0]][0], aNodes[ aTri[k][0]][1],
                aNodes[ aTri[k][2]][0], aNodes[ aTri[k][2]][1],QPen( QColor(0,255,0) ) );
            superScene->addLine(aNodes[ aTri[k][1]][0], aNodes[ aTri[k][1]][1],
                aNodes[ aTri[k][2]][0], aNodes[ aTri[k][2]][1],QPen( QColor(0,255,0) ) );
        }

}
double GraphWidget::areaOfTriangle(int Ax, int Ay, int Bx, int By, int Cx, int Cy)
{

    return (float) 0.5 * abs(Ax*By + Ay*Cx + Bx*Cy - By*Cx - Ax*Cy - Ay*Bx);

}


bool GraphWidget::insideTriangle(int Ax, int Ay, int Bx, int By, int Cx, int Cy, int Px, int Py)
{

    float areaInd,areaTotal; //areas

    areaInd = areaOfTriangle(Px,Py,Ax,Ay,Bx,By) + areaOfTriangle(Px,Py,Bx,By,Cx,Cy) + areaOfTriangle(Px,P
y,Cx,Cy,Ax,Ay);
    areaTotal = areaOfTriangle(Ax,Ay,Bx,By,Cx,Cy);
    if((areaInd == 0.0) and (areaTotal == 0.0)) printf("int Ax %d, int Ay%d, int Bx%d, int By%d int Px%d,
int Py%d\n",Ax,Ay,Bx,By,Px,Py);
    if ( ((areaInd - areaTotal)==0.0) && (areaInd != 0.0) and (areaTotal != 0.0) ) return true;
    else return false;

}
```

```
void GraphWidget::helpImageVisible(bool showit)
{

    QList<Node *> nodes;

    foreach (QGraphicsItem *item, scene()->items())

        if (Node *node = qgraphicsitem_cast<Node *>(item))
            if (node->helpNode == true) nodes << node;

        foreach (Node *node, nodes)
            node->setVisible(showit);

    faceReference->setVisible(showit);

    if (showit == true)
        scene()->setSceneRect(0,0, 760, 570);
    else
        scene()->setSceneRect(0,0, 380, 570);
}

// Master Thesis - StoneFace v.1.0 beta C by Jorge Garcia Bueno
// Universidad Carlos III de Madrid + University of Glasgow 2009
// All rights reserved by the author
// ---------------------------------------------------------------
// main.cpp - Starting the application...


#include <QApplication>

#include "mainwindow.h"

int main(int argc, char *argv[])
{

    // Creation of an application
    //Q_INIT_RESOURCE(application);
    QApplication app(argc, argv);

    // Show the main form
    MainWindow mainWin;
    mainWin.show();
    return app.exec();
}

// Master Thesis - StoneFace v.1.0 beta C by Jorge Garcia Bueno
// Universidad Carlos III de Madrid + University of Glasgow 2009
// All rights reserved by the author
// ---------------------------------------------------------------
// mainwindow.h - Header with the instructions for mainwindow.cpp

#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QtGui>

#include "graphwidget.h"
#include "mixer.h"
#include "cameraCapture.h"
#include "welcomewidget.h"
#include "automaticwidget.h"
#include "searchwidget.h"


class QAction;
class QMenu;
class QGraphWidget;
class WelcomeWidget;
class MainWindow : public QMainWindow
{

   Q_OBJECT

public:
    MainWindow();


public slots:
    void welcomeButton();

protected:
    void closeEvent(QCloseEvent *event);

private slots:
    void open();
    bool saveAs();
    void about();
    void average();
    void warpImage();
    void showLines(bool);
    void nextImage();
    void closePicture();
    void previousImage();

    void cameraCap();
    void showHelpFace(bool);

private:
    void createGeneralActions();
    void createGraphActions();
    void createMenus();
    void createToolBars();
    void createStatusBar();
    void readSettings();
    void writeSettings();
    bool maybeSave();
    void createWelcomeImage();
    void MoveCenterScreen();
    void createBackground();

    void loadFile(const QString &fileName);
    bool saveFile(const QString &fileName);
    void setCurrentFile(const QString &fileName);

    QPicture *background;
    QPainter *painterBack;

    QString strippedName(const QString &fullFileName);
    GraphWidget *widget;

    QString curFile;
    QStringList files;
    QStringList::Iterator it;

    QMenu *fileMenu;
    QMenu *viewMenu;
    QMenu *opMenu;
    QMenu *actionsMenu;
    QMenu *imgsMenu;
    QMenu *helpMenu;

    // Submenus
    QMenu *oneNode;
    QMenu *groupNodes;
    QMenu *allNodes;

    QToolBar *fileToolBar;
    QToolBar *editToolBar;
    QToolBar *nodeToolBar;
    QToolBar *groupToolBar;
    QToolBar *allToolBar;
    QToolBar *progressToolBar;
    QToolBar *viewToolBar;

    QWidget *backImage;

    QDockWidget *dock;

    QAction *openAct;
    QAction *saveAsAct;
    QAction *closeAct;
    QAction *avgAct;
    QAction *cameraAct;
    QAction *showHelpFaceAct;

    QAction *displaceNodeLeftAct;
    QAction *displaceNodeRightAct;
    QAction *displaceNodeUpAct;
    QAction *displaceNodeDownAct;
    QAction *displaceGroupLeftAct;
    QAction *displaceGroupRightAct;
    QAction *displaceGroupUpAct;
    QAction *displaceGroupDownAct;
    QAction *turnGroupLeftAct;
    QAction *turnGroupRightAct;
    QAction *increaseGroupDistanceAct;
    QAction *decreaseGroupDistanceAct;
    QAction *displaceAllLeftAct;
    QAction *displaceAllRightAct;
    QAction *displaceAllUpAct;
    QAction *displaceAllDownAct;

    QAction *showLinesAct;
    QAction *warpAct;
    QAction *nextAct;
    QAction *previousAct;

    QAction *exitAct;
    QAction *aboutAct;
    QAction *aboutQtAct;

    WelcomeWidget* ww;
    QProgressBar *pb;
    int numberOfImages;
    int contImages;
signals:

    // To manage communications with graphwidget

    // For one node
    void displaceNodeLeft();
    void displaceNodeRight();
    void displaceNodeUp();
    void displaceNodeDown();

    // For a group of nodes
    void displaceGroupLeft();
    void displaceGroupRight();
    void displaceGroupUp();
    void displaceGroupDown();

    void turnGroupLeft();
    void turnGroupRight();

    void increaseGroupDistance();
    void decreaseGroupDistance();

    // For the whole face
    void displaceAllLeft();
    void displaceAllRight();
    void displaceAllUp();
    void displaceAllDown();



};

#endif

// Master Thesis - StoneFace v.1.0 beta C by Jorge Garcia Bueno
// Universidad Carlos III de Madrid + University of Glasgow 2009
// All rights reserved by the author
// ------------------------------------------------------------
// mainwindow.cpp - Controls the application interface, windows and embedded pictures, lines, etc ...

#include "mainwindow.h"


MainWindow::MainWindow()
{


    // No file opened
    setCurrentFile("");
    createWelcomeImage();
    contImages = 0;
    widget = NULL;

}

void MainWindow::createGraphActions()
{

    connect(displaceNodeLeftAct, SIGNAL(triggered()), widget, SLOT(displaceNodeLeft()));
    connect(displaceNodeRightAct, SIGNAL(triggered()), widget, SLOT(displaceNodeRight()));
    connect(displaceNodeUpAct, SIGNAL(triggered()), widget, SLOT(displaceNodeUp()));
    connect(displaceNodeDownAct, SIGNAL(triggered()), widget, SLOT(displaceNodeDown()));

    connect(displaceGroupLeftAct, SIGNAL(triggered()), widget, SLOT(displaceGroupLeft()));
    connect(displaceGroupRightAct, SIGNAL(triggered()), widget, SLOT(displaceGroupRight()));
    connect(displaceGroupUpAct, SIGNAL(triggered()), widget, SLOT(displaceGroupUp()));
    connect(displaceGroupDownAct, SIGNAL(triggered()), widget, SLOT(displaceGroupDown()));
    connect(turnGroupRightAct, SIGNAL(triggered()), widget, SLOT(turnGroupRight()));
    connect(turnGroupLeftAct, SIGNAL(triggered()), widget, SLOT(turnGroupLeft()));
    connect(increaseGroupDistanceAct, SIGNAL(triggered()), widget, SLOT(increaseGroupDistance()));
    connect(increaseGroupDistanceAct, SIGNAL(triggered()), widget, SLOT(increaseGroupDistance()));


    connect(displaceAllLeftAct, SIGNAL(triggered()), widget, SLOT(displaceAllLeft()));
    connect(displaceAllRightAct, SIGNAL(triggered()), widget, SLOT(displaceAllRight()));
    connect(displaceAllUpAct, SIGNAL(triggered()), widget, SLOT(displaceAllUp()));
    connect(displaceAllDownAct, SIGNAL(triggered()), widget, SLOT(displaceAllDown()));

    connect(increaseGroupDistanceAct, SIGNAL(triggered()), widget, SLOT(increaseGroupDistance()));
    connect(decreaseGroupDistanceAct, SIGNAL(triggered()), widget, SLOT(decreaseGroupDistance()));




}
void MainWindow::createGeneralActions()
{

    openAct = new QAction(QIcon("./images/open.png"), tr("&Open..."), this);
    openAct->setShortcut(tr("Ctrl+O"));
    openAct->setStatusTip(tr("Open an existing picture"));
    connect(openAct, SIGNAL(triggered()), this, SLOT(open()));
    saveAsAct = new QAction(QIcon("./images/save.png"), tr("&Save as..."), this);
    saveAsAct->setShortcut(tr("Ctrl+S"));
    saveAsAct->setStatusTip(tr("Save the warped picture under a new name"));
    connect(saveAsAct, SIGNAL(triggered()), this, SLOT(saveAs()));

    closeAct = new QAction(QIcon("./images/close.png"), tr("&Close..."), this);
    closeAct->setShortcut(tr("Ctrl+X"));
    closeAct->setStatusTip(tr("Close actual work without saving changes"));
    connect(closeAct, SIGNAL(triggered()), this, SLOT(closePicture()));


    avgAct = new QAction(QIcon("./images/average.png"), tr("&Create average..."), this);
    avgAct->setShortcut(tr("Ctrl+A"));
    avgAct->setStatusTip(tr("Create the average image from multiple sources"));
    connect(avgAct, SIGNAL(triggered()), this, SLOT(average()));

    cameraAct = new QAction(QIcon("./images/camera.png"), tr("&Capture image from webcam..."), this);
    cameraAct->setShortcut(tr("Ctrl+W"));
    cameraAct->setStatusTip(tr("Captures a picture from the webcam to be used later on"));
    connect(cameraAct, SIGNAL(triggered()), this, SLOT(cameraCap()));

    showLinesAct = new QAction( QIcon("./images/lines.png"), tr("&Show mesh lines"), this);
    showLinesAct->setCheckable(true);
    showLinesAct->setChecked(false);
    showLinesAct->setShortcut(tr("Ctrl+L"));
    showLinesAct->setStatusTip(tr("Show all the conections between nodes"));
    connect(showLinesAct, SIGNAL(toggled(bool)), this, SLOT(showLines(bool)));


    showHelpFaceAct = new QAction(QIcon("./images/faceview.png"), tr("&Show help face on the right"), thi
s);
    showHelpFaceAct->setShortcut(tr("Ctrl+J"));
    showHelpFaceAct->setStatusTip(tr("Show the image face of the right with the help nodes"));
    showHelpFaceAct->setCheckable(true);
    showHelpFaceAct->setChecked(true);
    connect(showHelpFaceAct, SIGNAL(toggled(bool)), this, SLOT(showHelpFace(bool)));

    displaceNodeLeftAct = new QAction(QIcon("./images/left.png"), tr("Displace node to the left"), this);
    displaceNodeLeftAct->setStatusTip(tr("Move a node independently to the rest 5 pixels to the left"));

    displaceNodeRightAct = new QAction(QIcon("./images/right.png"), tr("Displace node to the rigth"), thi
s);
    displaceNodeRightAct->setStatusTip(tr("Move a node independently to the rest 5 pixels to the right"));

    displaceNodeUpAct = new QAction(QIcon("./images/up.png"), tr("Displace node up"), this);
    displaceNodeUpAct->setStatusTip(tr("Move a node independently to the rest 5 pixels up"));

    displaceNodeDownAct = new QAction(QIcon("./images/down.png"), tr("Displace node down"), this);
    displaceNodeDownAct->setStatusTip(tr("Move a node independently to the rest 5 pixels down"));

    displaceGroupLeftAct = new QAction(QIcon("./images/gleft.png"), tr("Displace selected group to the lef
t"), this);
    displaceGroupLeftAct->setStatusTip(tr("Move the group of nodes selected 5 pixels to the left"));

    displaceGroupRightAct = new QAction(QIcon("./images/gright.png"), tr("Displace selected group to the r
ight"), this);
    displaceGroupLeftAct->setStatusTip(tr("Move the group of nodes selected 5 pixels to the right"));

    displaceGroupUpAct = new QAction(QIcon("./images/gup.png"), tr("Displace selected group up"), this);
    displaceGroupUpAct->setStatusTip(tr("Move the group of nodes selected 5 pixels up"));

    displaceGroupDownAct = new QAction(QIcon("./images/gdown.png"), tr("Displace selected group down"), th
is);
    displaceGroupDownAct->setStatusTip(tr("Move the group of nodes selected 5 pixels down"));

    turnGroupLeftAct = new QAction(QIcon("./images/gtl.png"), tr("Turn selected group anticlockwise"), thi
s);
    turnGroupLeftAct->setStatusTip(tr("Turn the group of nodes selected 10 degrees anticlockwise (left)
"));
```

```
    turnGroupRightAct = new QAction(QIcon("./images/gtr.png"), tr("Turn selected group clockwise"), this);
    turnGroupRightAct->setStatusTip(tr("Turn the group of nodes selected 10 degrees clockwise (right)"));

    increaseGroupDistanceAct = new QAction(QIcon("./images/plus.png"), tr("Increase selected group dispers
ion"), this);
    increaseGroupDistanceAct->setStatusTip(tr("Increases the distances between the nodes of a selected gro
up"));

    decreaseGroupDistanceAct = new QAction(QIcon("./images/minus.png"), tr("Decrease selected group disper
sion"), this);
    decreaseGroupDistanceAct->setStatusTip(tr("Decreases the distances between the nodes of a selected gro
up"));

    displaceAllLeftAct  = new QAction(QIcon("./images/allleft.png"), tr("Displace all the mesh to the left
"), this);
    displaceAllLeftAct->setStatusTip(tr("Move all the nodes of the mesh 5 pixels to the left"));

    displaceAllRightAct  = new QAction(QIcon("./images/allright.png"), tr("Displace all the mesh to the ri
ght"), this);
    displaceAllRightAct->setStatusTip(tr("Move all the nodes of the mesh 5 pixels to the right"));

    displaceAllUpAct  = new QAction(QIcon("./images/allup.png"), tr("Displace all the mesh up"), this);
    displaceAllUpAct->setStatusTip(tr("Move all the nodes of the mesh 5 pixels up"));

    displaceAllDownAct  = new QAction(QIcon("./images/alldown.png"), tr("Displace all the mesh down"), thi
s);
    displaceAllDownAct->setStatusTip(tr("Move all the nodes of the mesh 5 pixels down"));

    warpAct = new QAction(QIcon("./images/warp.png"), tr("&Warp actual image"), this);
    warpAct->setStatusTip(tr("Apply warping algorithm to current image"));
    connect(warpAct, SIGNAL(triggered()), this, SLOT(warpImage()));

    nextAct = new QAction(QIcon("./images/next.png"), tr("&Continue with the next image of the list"), thi
s);
    nextAct->setStatusTip(tr("Change to the next image"));
    connect(nextAct, SIGNAL(triggered()), this, SLOT(nextImage()));

    previousAct = new QAction(QIcon("./images/previous.png"), tr("&Return to the previous image of the lis
t"), this);
    previousAct->setStatusTip(tr("Change to the previous image"));
    connect(previousAct, SIGNAL(triggered()), this, SLOT(previousImage()));


    exitAct = new QAction(tr("E&xit"), this);
    exitAct->setShortcut(tr("Ctrl+Q"));
    exitAct->setStatusTip(tr("Exit the application"));
    connect(exitAct, SIGNAL(triggered()), this, SLOT(close()));

    aboutAct = new QAction(tr("&About"), this);
    aboutAct->setStatusTip(tr("Show the application's About box"));
    connect(aboutAct, SIGNAL(triggered()), this, SLOT(about()));

    aboutQtAct = new QAction(tr("About &Qt"), this);
    aboutQtAct->setStatusTip(tr("Show the Qt library's About box"));
    connect(aboutQtAct, SIGNAL(triggered()), qApp, SLOT(aboutQt()));

}

void MainWindow::createMenus()
{
    fileMenu = menuBar()->addMenu(tr("&File"));
    fileMenu->addAction(openAct);
    fileMenu->addAction(saveAsAct);
    fileMenu->addSeparator();
    fileMenu->addAction(closeAct);
    fileMenu->addSeparator();
    fileMenu->addAction(exitAct);


    viewMenu = menuBar()->addMenu(tr("&View"));
    viewMenu->addAction(showLinesAct);
    viewMenu->addAction(showHelpFaceAct);

    opMenu = menuBar()->addMenu(tr("&Mesh..."));

    oneNode = opMenu->addMenu(tr("&Selected node"));
        oneNode->addAction(displaceNodeLeftAct);
        oneNode->addAction(displaceNodeRightAct);
        oneNode->addSeparator();
        oneNode->addAction(displaceNodeUpAct);
        oneNode->addAction(displaceNodeDownAct);

    groupNodes = opMenu->addMenu(tr("Selected &group of nodes"));
        groupNodes->addAction(displaceGroupLeftAct);
        groupNodes->addAction(displaceGroupRightAct);
        groupNodes->addSeparator();
        groupNodes->addAction(displaceGroupUpAct);
        groupNodes->addAction(displaceGroupDownAct);
        groupNodes->addSeparator();
        groupNodes->addAction(turnGroupLeftAct);
        groupNodes->addAction(turnGroupRightAct);
        groupNodes->addSeparator();
        groupNodes->addAction(increaseGroupDistanceAct);
        groupNodes->addAction(decreaseGroupDistanceAct);

    allNodes = opMenu->addMenu(tr("&All nodes of the mesh"));
        allNodes->addAction(displaceAllLeftAct);
        allNodes->addAction(displaceAllRightAct);
        allNodes->addAction(displaceAllUpAct);
        allNodes->addAction(displaceAllDownAct);


    actionsMenu = menuBar()->addMenu("&Operations");
        actionsMenu->addAction(avgAct);
        actionsMenu->addAction(warpAct);
        actionsMenu->addSeparator();
        actionsMenu->addAction(cameraAct);

    imgsMenu = menuBar()->addMenu(tr("&Images..."));
        imgsMenu->addAction(previousAct);
        imgsMenu->addAction(nextAct);

    helpMenu = menuBar()->addMenu(tr("&Help"));
        helpMenu->addAction(aboutAct);
        helpMenu->addAction(aboutQtAct);
}

void MainWindow::createToolBars()
{

    // First create the horizontal tool bar

    fileToolBar = addToolBar(tr("File"));
    fileToolBar->setIconSize(QSize(40,40));

    fileToolBar->addAction(openAct);
    fileToolBar->addAction(saveAsAct);


    editToolBar = addToolBar(tr("Operations"));
    editToolBar->setIconSize(QSize(40,40));

    editToolBar->addAction(warpAct);
    editToolBar->addAction(avgAct);
    editToolBar->addAction(cameraAct);
    editToolBar->addSeparator();
    editToolBar->addAction(previousAct);
    editToolBar->addAction(nextAct);

    viewToolBar = addToolBar(tr("Views"));
    viewToolBar->setIconSize(QSize(40,40));
    viewToolBar->addAction(showLinesAct);
    viewToolBar->addAction(showHelpFaceAct);

    pb = new QProgressBar(this);

    progressToolBar = addToolBar(tr("Progress"));
    progressToolBar->addWidget(pb);


    // Second creation of the 'dockable' vertical bar


    nodeToolBar = new QToolBar(tr("Individual nodes"), this);
    nodeToolBar->setIconSize(QSize(22,22));

    nodeToolBar->addAction(displaceNodeLeftAct);
    nodeToolBar->addAction(displaceNodeRightAct);
    nodeToolBar->addAction(displaceNodeUpAct);
    nodeToolBar->addAction(displaceNodeDownAct);

    groupToolBar = new QToolBar(tr("Group of nodes"), this);
    groupToolBar->setIconSize(QSize(22,22));

    groupToolBar->addAction(displaceGroupLeftAct);
    groupToolBar->addAction(displaceGroupRightAct);
    groupToolBar->addAction(displaceGroupUpAct);
    groupToolBar->addAction(displaceGroupDownAct);
    groupToolBar->addAction(turnGroupLeftAct);
    groupToolBar->addAction(turnGroupRightAct);
    groupToolBar->addAction(increaseGroupDistanceAct);
    groupToolBar->addAction(decreaseGroupDistanceAct);

    allToolBar = new QToolBar(tr("All nodes"), this);
    allToolBar->setIconSize(QSize(22,22));

    allToolBar->addAction(displaceAllLeftAct);
    allToolBar->addAction(displaceAllRightAct);
    allToolBar->addAction(displaceAllUpAct);
    allToolBar->addAction(displaceAllDownAct);

    addToolBar(Qt::LeftToolBarArea,nodeToolBar);
    addToolBar(Qt::LeftToolBarArea,groupToolBar);
    addToolBar(Qt::LeftToolBarArea,allToolBar);
}

void MainWindow::createStatusBar()
{
    statusBar()->showMessage(tr("Ready to warp! :)"));
}

void MainWindow::createWelcomeImage()
{
    setWindowTitle(tr("StoneFace v.1 - Select Option"));
    resize(QSize(3*128+80, 180));

    // Move the form to the center of the screen
    MoveCenterScreen();

    ww = new WelcomeWidget(this);
    setCentralWidget(ww);

}


void MainWindow::readSettings()
{
    QSettings settings("Warping", "Settings");
    QPoint pos = settings.value("pos", QPoint(200, 200)).toPoint();
    QSize size = settings.value("size", QSize(1000, 1000)).toSize();
    resize(size);
    move(pos);
}

void MainWindow::writeSettings()
{
    QSettings settings("Warping", "Settings");
    settings.setValue("pos", pos());
    settings.setValue("size", size());
}

void MainWindow::createBackground()
{


    backImage = new QWidget(this);
    QPixmap bg("./images/background.png");

    QPalette p(palette());
    p.setBrush(QPalette::Background, bg);
    setPalette(p);

    setCentralWidget(backImage);
    //repaint();
    repaint();
/*
    QStringList list = QFileDialog::getOpenFileNames(
                          0,
                          "Select one or more images to warp!",
                          "./",
                          "Images (*.png *.xpm *.jpg *.bmp *.gif)");

    // We store the files selected in a FileList object

    numberOfImages = list.size();

    it = list.begin();


    while (it != list.end())
    {
        QImage img(*it);
        img = img.scaled(90,90,Qt::IgnoreAspectRatio,Qt::SmoothTransformation);
        img.save(QString(*it).append("-mini.jpg"));


        it++;
    }
*/
}

void MainWindow::welcomeButton()
{
    ww->setVisible(false);

    switch (ww->itemSelected())
    {
        case 0:
        {
            // Design the interface of the main window
            createGeneralActions();
            createMenus();
            createToolBars();
            createStatusBar();
            // Paint a background image
            createBackground();
            // Remember last settings (width, height, top, left)
            readSettings();
            MoveCenterScreen();
            setWindowTitle(tr("StoneFace v.1 - Manual Face Creator"));
```

```
                break;
        }
        case 1:
        {
            // Manage the widget position and properties of the window
            resize(QSize(3*180, 3*180));
            MoveCenterScreen();
            createStatusBar();
            setWindowTitle(tr("StoneFace v.1 - Automatic Face Creator"));
            AutomaticWidget *auw = new AutomaticWidget(this);
            setCentralWidget(auw);

            break;
        }
        case 2:
        {
            setWindowTitle(tr("StoneFace v.1 - Face Search Engine"));
            SearchWidget *sw = new SearchWidget(this);
            MoveCenterScreen();
            setCentralWidget(sw);
            MoveCenterScreen();

            break;
        }

    }
}

void MainWindow::closeEvent(QCloseEvent *event)
{
    QMessageBox msgBox(this);
    msgBox.setText("Are you sure you want to leave the actual work and exit?");
    msgBox.setWindowTitle("Exiting!");
    msgBox.setIconPixmap(QPixmap("./images/close.png"));
    msgBox.setStandardButtons(QMessageBox::Yes | QMessageBox::No);

    int ret = msgBox.exec();

    if (ret == QMessageBox::Yes)
        {
            writeSettings();
            event->accept();
        }
    else
        {
            event->ignore();
        }
}


void MainWindow::open()
{

    // Let the user select the number of images desired
    // File dialog to select them manually with multiple selection enabled

    QStringList list = QFileDialog::getOpenFileNames(
                        0,
                        "Select one or more images to warp!",
                        "./",
                        "Images (*.png *.xpm *.jpg *.bmp *.gif)");

    // We store the files selected in a FileList object
    files = list;
    numberOfImages = list.size();

    pb->setMaximum(numberOfImages);
    pb->setMinimum(0);

    //Creation of a iterator to move inside the list
    it = files.begin();

    // Now we start one by one managing the warpings
    // First we present the user the first of the pictures list
    // Once it has finished with it (and press warp and next)
    // The following image is present and the values are restored

    if (it != files.end())
        loadFile(*it);

    // Now we let the user play with the image and we wait until 'next' is pressed

/* Old School opening mode
    //if (maybeSave()) {
        QString fileName = QFileDialog::getOpenFileName(this);
        if (!fileName.isEmpty())

   //}
*/
}


bool MainWindow::saveAs()
{

    if (widget != NULL)
    {
        QString fileName = QFileDialog::getSaveFileName(this);
        if (fileName.isEmpty())
                return false;

            return saveFile(fileName);
    }

    return false;
}

void MainWindow::closePicture()
{

    if (widget != NULL)
    {

        QMessageBox msgBox(this);
        msgBox.setText("Are you sure you want to remove actual work without saving?");
        msgBox.setInformativeText("");
        msgBox.setStandardButtons(QMessageBox::Yes | QMessageBox::No);
        msgBox.setIconPixmap(QPixmap("./images/Exclamation.png"));
        int ret = msgBox.exec();

        if (ret == QMessageBox::Yes)
        {
            delete widget;
            contImages = 0;
            createBackground();
        }
    }
}
void MainWindow::about()
{
   QMessageBox::about(this, tr("About Application"),
            tr("The <b>StoneFace v.1</b> This application was developed to generate averaging faces "
               "in order to be used to boost the actual techniques of face recognition.\n"
               "<b>University of Glasgow</b> - 2009\n<b>Jorge Garcia Bueno</b>"));
}

void MainWindow::showLines(bool showOrNot)
{
Mixer *m = new Mixer(NULL);
m->simpleAverage2(false);
```

```
    if (widget != NULL)
    {

        if (showOrNot == true)
        {
                widget->printEdges();

            statusBar()->showMessage(tr("Mesh printed succesfully"), 2000);

        }
        else
        {

            widget->removeEdges();

        }

    }

}

void MainWindow::previousImage()
{


    if (widget != NULL)
    {

        if (it != files.begin())
        {
            it--;
            contImages--;
            QApplication::setOverrideCursor(Qt::WaitCursor);
            widget->updatePicture(false);
            widget->savePicture(QString (""));
            QApplication::restoreOverrideCursor();
            QApplication::setOverrideCursor(Qt::WaitCursor);
            loadFile(*it);
            QApplication::restoreOverrideCursor();

        }
    }




}
void MainWindow::nextImage()
{

    if (widget != NULL)
    {
        // Advance one position in the list
        it++;
        contImages++;

        // Save the warped picture and the mesh
        QApplication::setOverrideCursor(Qt::WaitCursor);
        widget->updatePicture(false);
        widget->savePicture(QString (""));
        QApplication::restoreOverrideCursor();

        // Check there are more in the list
        if (it != files.end())
        {
            QApplication::setOverrideCursor(Qt::WaitCursor);

            loadFile(*it);
            QApplication::restoreOverrideCursor();
        }
        else
        {
            QMessageBox msgBox(this);
            QAbstractButton *openMoreButton = msgBox.addButton(tr("Create average..."), QMessageBox::Actio
nRole);
            QAbstractButton *CancelButton = msgBox.addButton(tr("No, thanks"), QMessageBox::ActionRole);
            msgBox.setText("Congratulations! You have reached the end of the list of pictures!");
            msgBox.setIcon(QMessageBox::Information);
            msgBox.exec();

            if (msgBox.clickedButton() == openMoreButton)
            {
                average();
            }
            if (msgBox.clickedButton() == CancelButton)
            {
                delete widget;
                // exit?¿
            }
        }
    }
}





bool MainWindow::maybeSave()
{
    /*if (textEdit->document()->isModified()) {
        QMessageBox::StandardButton ret;
        ret = QMessageBox::warning(this, tr("Application"),
                     tr("The picture has been modified.\n"
                        "Do you want to save your changes?"),
                      QMessageBox::Save | QMessageBox::Discard | QMessageBox::Cancel);
        if (ret == QMessageBox::Save)
            return save();
        else if (ret == QMessageBox::Cancel)
            return false;
    }*/
    return true;
}


void MainWindow::cameraCap()
{
    cameraCapture *camera = new cameraCapture();
    camera->captureWindow();
}



void MainWindow::loadFile(const QString &fileName)
{
    QFile file(fileName);

    if (!file.open(QFile::ReadOnly | QFile::Text))
    {
        QMessageBox::warning(this, tr("Warp Faces"),
                             tr("Cannot read file %1:\n%2.")
                             .arg(fileName)
                             .arg(file.errorString()));
        return;
    }

    QApplication::setOverrideCursor(Qt::WaitCursor);
    // Update the  progressbar

    pb->setValue(contImages);

    // Starting of a new widget
    widget = new GraphWidget();


    setCentralWidget(widget);
    createGraphActions();
    // Before loading the picture check if there is an old mesh and ask
    // if the user want to load it or not if found

    QString fileNameMesh;
    fileNameMesh.insert(0,fileName.mid(0,fileName.length()-4));
    fileNameMesh = fileNameMesh.insert(fileName.length()-4,"-warped.dat");

    QFile fileMesh(fileNameMesh);

    if (fileMesh.open(QFile::ReadOnly | QFile::Text))
    {
        // There is a previous mesh

        QMessageBox msgBox(this);
        msgBox.setText(tr("There is a previous mesh for this image, do you want to load it?"));
        msgBox.setInformativeText(tr("Previous mesh found!"));
        msgBox.setIcon(QMessageBox::Information);
        msgBox.setStandardButtons(QMessageBox::Yes | QMessageBox::No);

        int ret = msgBox.exec();

        if (ret == QMessageBox::Yes)
            fileNameMesh = fileNameMesh;
        else
            fileNameMesh = QString("pos.dat");

    }
    else
        fileNameMesh = QString("pos.dat");

    widget->loadPicture(fileName,fileNameMesh.toLatin1().data(),(char*)"pos.dat",false);

    QApplication::restoreOverrideCursor();
    setCurrentFile(fileName);
    statusBar()->showMessage(tr("File loaded: ").append(fileName), 2000);

}



bool MainWindow::saveFile(const QString &fileName)
{
    QFile file(fileName);
    if (!file.open(QFile::WriteOnly | QFile::Text)) {
        QMessageBox::warning(this, tr("Application"),
                             tr("Cannot write file %1:\n%2.")
                             .arg(fileName)
                             .arg(file.errorString()));
        return false;
    }

    //QTextStream out(&file);
    QApplication::setOverrideCursor(Qt::WaitCursor);

    widget->savePicture(fileName);
    QApplication::restoreOverrideCursor();
    setCurrentFile(fileName);
    statusBar()->showMessage(tr("File saved:").append(fileName), 2000);
    return true;
}
```

```
void MainWindow::warpImage()
{
    if (widget != NULL)
        widget->updatePicture(true);
}



void MainWindow::average()
{

    Mixer *mixer = new Mixer(widget);
    mixer->simpleAverage(false);
    statusBar()->showMessage(tr("Average created successfully"), 2000);
}
void MainWindow::setCurrentFile(const QString &fileName)
{
    curFile = fileName;
    //textEdit->document()->setModified(false);
    setWindowModified(false);

    QString shownName;
    if (curFile.isEmpty())
        shownName = "";
    else
        shownName = strippedName(curFile);

    setWindowTitle(tr("StoneFace v.1 - Main Menu"));
}

QString MainWindow::strippedName(const QString &fullFileName)
{
    return QFileInfo(fullFileName).fileName();
}
void MainWindow::MoveCenterScreen()
{

    int screenWidth, width;
    int screenHeight, height;


    QDesktopWidget *desktop = new QDesktopWidget();
    QSize windowSize;
    screenWidth = desktop->width(); // get width of screen
    screenHeight = desktop->height(); // get height of screen

    windowSize = size(); // size of our application window
    width = windowSize.width();
    height = windowSize.height();

    move((screenWidth-width)/2,(screenHeight - height)/2);
}

void MainWindow::showHelpFace(bool showit)
{
   if (widget != NULL)
        widget->helpImageVisible(showit);

}

// Master Thesis - StoneFace v.1.0 beta C by Jorge Garcia Bueno
// Universidad Carlos III de Madrid + University of Glasgow 2009
// All rights reserved by the author
// ---------------------------------------------------------------
// mixer.h - Header with the instructions for mixer.cpp


#ifndef MIXER_H
#define MIXER_H
```

```
// Include header files
#include "graphwidget.h"
#include <QtGui/QGraphicsView>
#include <QFileDialog>
#include <QMessageBox>
 class QAction;
 class QMenu;
 class QGraphWidget;


class Mixer
{

private:
     GraphWidget *widget;
public:
    // Constructor of the object
    Mixer(GraphWidget *);
    void autoDetectAverage();
    int simpleAverage(bool);
    int simpleAverage2(bool);
    void equalizeHistogram(uchar *, int, int);
    void HighPassFilter(uchar *pixels,int width, int height);
};



#endif

// Master Thesis - StoneFace v.1.0 beta C by Jorge Garcia Bueno
// Universidad Carlos III de Madrid + University of Glasgow 2009
// All rights reserved by the author
// ---------------------------------------------------------------
// mixer.cpp - Averaging algorithm to create a final mask with the individual warps when manual mode is es
tablished

#include "mixer.h"

Mixer::Mixer(GraphWidget *mainw)
{
    widget = mainw;

}

void Mixer::autoDetectAverage()
{
    QStringList files = QFileDialog::getOpenFileNames(
                         0,
                         "Select one or more files to open",
                         "./",
                         "Images (*.png *.xpm *.jpg *.bmp)");


    QStringList list = files;
    QStringList::Iterator it = list.begin();

    QImage *avgPic = new QImage(380,570,QImage::Format_RGB32);
    QList<qreal> avgMesh;

    unsigned int avgPicArray[3*380][570];
    int r,g,b;
    QColor *co;
    int k = 0,nTotal=list.size(),nRejected=0;

    // Reset the image matrix with 0-values

    for (int i=0;i<570;i++)
      for (int j=0;j<380*3;j++)
```

```
        avgPicArray[j][i]=0;

    // Reset the average array with 0-values
    for (int i = 0; i < 164; i++) avgMesh.append(0);
    avgMesh.append(0); avgMesh.append(0);
    avgMesh.append(380); avgMesh.append(0);
    avgMesh.append(380); avgMesh.append(570);
    avgMesh.append(0); avgMesh.append(570);
    //Necesary to check that all the images have the same size ...


    // Start iterations
    while( it != list.end() )
    {

        widget = new GraphWidget();
        // We will reject the picture if any of the elements of the face is NOT found
        if (( widget->loadPicture(*it,(char*)"pos.dat",(char*)"pos.dat",true) == 0) )
        {
        // Save the warped picture and the mesh

        widget->updatePicture(true);
        widget->savePicture(QString (""));

        widget->downloadAddMesh(&avgMesh);
        //printf("value element 10 avgMesh %f %f\n",avgMesh.value(10),avgMesh.value(11));
        for (int i=0;i<570;i++)
        {
          for (int j=0;j<380*3;j=j+3)
          {
            co = new QColor(widget->currentImage().pixel(k++,i));

            co->getRgb(&r,&g,&b);
            avgPicArray[j+0][i] += r;
            avgPicArray[j+1][i] += g;
            avgPicArray[j+2][i] += b;

          }k=0;
        }
        ++it;
        }

        else
        {
            nTotal--;
            printf("Picture rejected!\n");
            ++it;
        }

    }


    k=0;

//Average of the image, the vector and set the pixel
for (int i=0;i<570;i++)
{
  for (int j=0;j<380*3;j=j+3)
    {

        avgPicArray[j][i]   = (unsigned char)(avgPicArray[j+0][i]/nTotal);
        avgPicArray[j+1][i] = (unsigned char)(avgPicArray[j+1][i]/nTotal);
        avgPicArray[j+2][i] = (unsigned char)(avgPicArray[j+2][i]/nTotal);

        co = new QColor(avgPicArray[j][i],avgPicArray[j+1][i],avgPicArray[j+2][i]);
        avgPic->setPixel(k++,i,co->rgb());
    }k=0;
}
```

```
// Update the average array with the new values
for (int i = 0;i < 164;i++)avgMesh.replace(i,avgMesh.value(i) / list.size()-nRejected);


// Save the average mesh in a file to be sent to the widget object
    FILE* f=fopen("average.dat","w");

    if (!f) {
        fprintf(stderr,"Error saving the mesh file!\n");
        exit(1);
    }
    fprintf(f,"86\n");

    for (int i = 0;i < 172;i = i +2)
    fprintf(f,"%d %d b\n",(int)avgMesh.value(i),(int)avgMesh.value(i+1));


// Now we need to deform back the picture to the avg mesh
/*
        widget = new GraphWidget();

        widget->loadPicture("average.jpg","pos.dat","avg.dat",false);
        widget->updatePicture();
        widget->savePicture("morphedAverage.jpg");
*/
    // Save the average picture
    avgPic->save("average.bmp");
    QMessageBox msgBox;
    msgBox.setText("Image created and saved succesfully: average.bmp\n");
    msgBox.exec();
    exit(0);



}



int Mixer::simpleAverage(bool equalization)
{

    QStringList files = QFileDialog::getOpenFileNames(
                         0,
                         "Select one or more files to open",
                         "./",
                         "Images (*.png *.xpm *.jpg *.bmp)");

    QStringList::Iterator it = files.begin();
    QString s;
    QImage resImg ;

    QImage avgPic(380,570,QImage::Format_RGB32);


    int nNodes = 0,x,y;
    int cont = 0;
    char color;
    QString fileNameMesh;

    int r,g,b;
    QColor co;
    int k = 0, z;
    int nTotal = files.size();


    if (nTotal == 0) return (0);
    // Reserve memory to do the process
    double avgMesh[172];
    unsigned long avgPicArray[3*380][570];
```

```
// Reset the average matrix with 0-values
for (int i=0;i<570;i++)
  for (int j=0;j<380*3;j++)
    avgPicArray[j][i]=0;


// Create the list and the array with 0 values
for (int i = 0; i < 172; i++)
    *(avgMesh + i) = 0;

while( it != files.end() )
{

    // We will not reject any picture because there is not pre-allocating
    avgPic.load(s);

    // Edit the name of the mesh file associated with the image file
    fileNameMesh = QString(*it);
    fileNameMesh.chop(3);
    fileNameMesh.append(QString("dat"));
    s = *it;


    fileNameMesh = QString(s);
    fileNameMesh.chop(3);
    fileNameMesh.append(QString("dat"));


    // Read the file mesh values to store them into avgMesh
    FILE* f=fopen(fileNameMesh.toLatin1().data(),"r");
      if (!f)
       {
            fprintf(stderr,"Warped mesh: %s is missing!\n",fileNameMesh.toLatin1().data());
            exit(1);
       }

    // Load values from the mesh file
    z = fscanf(f,"%d\n",&nNodes);

    for(int i=0; i<nNodes; i++)
    {
        z = fscanf(f,"%d %d %c\n",&x,&y,&color);

        *(avgMesh + cont)    = *(avgMesh + cont)     + (double)x ;
        *(avgMesh + cont + 1) =*( avgMesh + cont+1)  + (double)y ;
        cont = cont + 2;
    }
    cont = 0;

    fclose(f);

    // Store the values into avgPicArray
    for (int i=0;i<570;i++)
    {
      for (int j=0;j<380*3;j=j+3)
      {
        co = QColor(avgPic.pixel(k++,i));

        co.getRgb(&r,&g,&b);
        avgPicArray[j+0][i] += r;
        avgPicArray[j+1][i] += g;
        avgPicArray[j+2][i] += b;

      }k=0;
    }

    //printf("ruta procesada: %s\n", s.toLatin1().data());

    it++;
    }

    k = 0;

    //Average of the image, the vector and set the pixel
    for (int i=0;i<570;i++)
    {
      for (int j=0;j<380*3;j=j+3)
        {

            avgPicArray[j+0][i] = (unsigned char)(avgPicArray[j+0][i]/nTotal);
            avgPicArray[j+1][i] = (unsigned char)(avgPicArray[j+1][i]/nTotal);
            avgPicArray[j+2][i] = (unsigned char)(avgPicArray[j+2][i]/nTotal);

            co = QColor(avgPicArray[j][i],avgPicArray[j+1][i],avgPicArray[j+2][i]);
            avgPic.setPixel(k++,i,co.rgb());
        }k=0;
    }

    // Division of all the values by the number of pictures
    for (int i = 0;i < 172;i++)
        avgMesh[i]=avgMesh[i] / nTotal;


   // Save the pre-made picture to process the last part
   avgPic.save("/tmp/average.jpg");


    // Save the average mesh in a file
    FILE* ff=fopen("avg.dat","w");

    if (!ff) {
        fprintf(stderr,"Error saving the mesh file!\n");
        exit(1);
    }
    //Print the number of nodes first
    fprintf(ff,"86\n");


    for (int i = 0;i < 172;i = i + 2)
        fprintf(ff,"%d %d g\n",(int)(avgMesh[i]), (int) (avgMesh[i+1]) );

    fclose(ff);

    //Finally we need to deform back the picture to the avg mesh
    QString avgee = QFileDialog::getSaveFileName(NULL,
                    QString("Save Image"), QString("./"),QString("Image Files (*.png *.jpg *.bmp)"));

    widget = new GraphWidget();

    widget->loadPicture("/tmp/average.jpg",(char*)"pos.dat",(char*)"avg.dat",false);
    widget->updatePicture(false);
    //widget->savePicture(avgee);

    avgPic =  QImage(widget->currentImage());
    resImg = avgPic.scaled(190,285,Qt::IgnoreAspectRatio,Qt::SmoothTransformation);
    //if (equalization == true) equalizeHistogram(avgPic.bits(),190,285);
    resImg.save(avgee);

return (1);

}

void Mixer::equalizeHistogram(uchar *pixels,int width, int height)
{

// First of all obtain the max and min values of all the range of grays in the picture
uchar min_r=255,min_g=255,min_b=255;
uchar max_r=0,max_g=0,max_b=0;
uchar R,G,B;
```

```
for (int i = 0;i < 3 * width * height;i=i+3)
{

    *(pixels+i+0)=(char)255 * ( ( 1 + *(pixels+i+0) ) / log(256));
    *(pixels+i+1)=(char)255 * ( ( 1 + *(pixels+i+1) ) / log(256));
    *(pixels+i+2)=(char)255 * ( ( 1 + *(pixels+i+2) ) / log(256));

    R = *(pixels+i+0);
    G = *(pixels+i+1);
    B = *(pixels+i+2);

    if (B  > max_b) max_b = B;
    if (B  < min_b) min_b = B;
    if (R  > max_r) max_r = R;
    if (R  < min_r) min_r = R;
    if (G  > max_g) max_g = G;
    if (G  < min_g) min_g = G;
}

// Reescalation of all the pixels
for (int i = 0;i < 3 * width * height;i=i+3)
{
    *(pixels+i+0) = (*(pixels+i+0)-min_r)*255/(max_r-min_r);
    *(pixels+i+1) = (*(pixels+i+1)-min_g)*255/(max_g-min_g);
    *(pixels+i+2) = (*(pixels+i+2)-min_b)*255/(max_b-min_b);
}

}

void Mixer::HighPassFilter(uchar *pixels,int width, int height)
{


char mask[3][3] ={{0,-1,0},
               {-1,5,-1},
               {0,-1,0}};
IplImage* img = cvCreateImage(cvSize(width, height), IPL_DEPTH_8U, 3);
char newRed, newBlue, newGreen;
char *pixFiltered;
cvGetRawData(img, (uchar**)&pixFiltered);
int k=0;
for (int i = 3 + width;i < 3 * width * (height - 1) - 3 ;i=i+3)
{

    newRed =(char)(1/9.0)*(mask[0][0] * *(pixels + i - width - 3) +
                          mask[0][1] * *(pixels + i - width + 0) +
                          mask[0][2] * *(pixels + i - width + 3) +
                          mask[1][0] * *(pixels + i - 3)         +
                          mask[1][1] * *(pixels + i + 0)         +
                          mask[1][2] * *(pixels + i + 3)         +
                          mask[2][0] * *(pixels + i + width - 3) +
                          mask[2][1] * *(pixels + i + width + 0) +
                          mask[2][2] * *(pixels + i + width + 3));

    newGreen=(char)(1/9.0)*(mask[0][0] * *(pixels + i - width - 2) +
                          mask[0][1] * *(pixels + i - width + 1) +
                          mask[0][2] * *(pixels + i - width + 4) +
                          mask[1][0] * *(pixels + i - 2)         +
                          mask[1][1] * *(pixels + i + 1)         +
                          mask[1][2] * *(pixels + i + 4)         +
                          mask[2][0] * *(pixels + i + width - 2) +
                          mask[2][1] * *(pixels + i + width + 1) +
                          mask[2][2] * *(pixels + i + width + 4));

    newBlue = (char)(1 /9.0) *( mask[0][0] * *(pixels + i - width - 1) +
                          mask[0][1] * *(pixels + i - width + 2) +
                          mask[0][2] * *(pixels + i - width + 5) +
                          mask[1][0] * *(pixels + i - 1)         +
                          mask[1][1] * *(pixels + i + 2)         +
                          mask[1][2] * *(pixels + i + 5)         +
                          mask[2][0] * *(pixels + i + width - 1) +
                          mask[2][1] * *(pixels + i + width + 2) +
                          mask[2][2] * *(pixels + i + width + 5));

    * (pixFiltered + k + 0) = newRed;
    * (pixFiltered + k + 1) = newGreen;
    * (pixFiltered + k + 2) = newBlue;

    k++;
}


for (int i = 0;i < 3 * width * height;i=i+3)
{


    *(pixels+i+0)=(char)* (pixFiltered + i + 0);
    *(pixels+i+1)=(char)* (pixFiltered + i + 1);
    *(pixels+i+2)=(char)* (pixFiltered + i + 2);

}


}






int Mixer::simpleAverage2(bool equalization)
{

    QStringList files = QFileDialog::getOpenFileNames(
                         0,
                         "Select one or more files to open",
                         "./",
                         "Images (*.png *.xpm *.jpg *.bmp)");

    QStringList::Iterator it = files.begin();
    QString s;
    QImage resImg ;

    QImage avgPic(380,570,QImage::Format_RGB32);


    int nNodes = 0,x,y;
    int cont = 0;
    char color;
    QString fileNameMesh;

    int r,g,b;
    QColor co;
    int k = 0, z;
    int nTotal = files.size();


    if (nTotal == 0) return (0);
    // Reserve memory to do the process

    unsigned long avgPicArray[3*380][570];

    // Reset the average matrix with 0-values
    for (int i=0;i<570;i++)
      for (int j=0;j<380*3;j++)
        avgPicArray[j][i]=0;


    while( it != files.end() )
    {
        // We will not reject any picture because there is not pre-allocating
        s = *it;
        avgPic.load(s);
        avgPic = avgPic.scaled(380,570,Qt::IgnoreAspectRatio,Qt::SmoothTransformation);
        // Store the values into avgPicArray
        for (int i=0;i<570;i++)
        {
          for (int j=0;j<380*3;j=j+3)
          {
            co = QColor(avgPic.pixel(k++,i));

            co.getRgb(&r,&g,&b);
            avgPicArray[j+0][i] += r;
            avgPicArray[j+1][i] += g;
            avgPicArray[j+2][i] += b;

          }k=0;
        }

        //printf("ruta procesada: %s\n", s.toLatin1().data());

        it++;
    }

    // Division of all the values by the number of pictures
        for (int i=0;i<570;i++)
        {
          for (int j=0;j<380*3;j=j+3)
          {

            avgPicArray[j+0][i] = avgPicArray[j+0][i]/nTotal;
            avgPicArray[j+1][i] = avgPicArray[j+1][i]/nTotal;
            avgPicArray[j+2][i] = avgPicArray[j+2][i]/nTotal;

          }k=0;
        }

    k = 0;

    //Average of the image, the vector and set the pixel
    for (int i=0;i<570;i++)
    {
      for (int j=0;j<380*3;j=j+3)
        {

            co = QColor(avgPicArray[j][i],avgPicArray[j+1][i],avgPicArray[j+2][i]);
            avgPic.setPixel(k++,i,co.rgb());
        }k=0;
    }

    //Finally we need to deform back the picture to the avg mesh
    QString avgee = QFileDialog::getSaveFileName(NULL,
                    QString("Save Image"), QString("./"),QString("Image Files (*.png *.jpg *.bmp)"));
    avgPic = avgPic.scaled(190,285,Qt::IgnoreAspectRatio,Qt::SmoothTransformation);
    avgPic.save(avgee);

return (1);

}


// Master Thesis - StoneFace v.1.0 beta C by Jorge Garcia Bueno
// Universidad Carlos III de Madrid + University of Glasgow 2009
// All rights reserved by the author
// ---------------------------------------------------------------
// mixeraut.h - Header with the instructions for mixeraut.cpp

#ifndef MIXERAUT_H
#define MIXERAUT_H
```

```
// Include header files

#include <QtGui>
#include <QPixmap>
#include <QGraphicsView>
#include <QListView>
#include <QDialog>
#include <QPushButton>
#include <QDesktopWidget>
#include "stonefacewidget.h"
#include "automaticwidget.h"


class QAction;
class QMenu;
class QGraphWidget;
class WelcomeWidget;
class AutomaticWidget;
class StoneFaceWidget;
class MixerAutomatic : public QWidget
{
    Q_OBJECT

public:
    // Constructor of the object
    MixerAutomatic(StoneFaceWidget *, AutomaticWidget *);
    int Start(QString,int,int,bool);
    int setFilesList(QStringList);
private:

    AutomaticWidget *qw;
    StoneFaceWidget *widget;
    QStringList *files;

signals:
void SetStatus(QString,int,int,int,int,int,int,int, QImage, QImage);


public slots:
    int SearchFiles();

private slots:

void PBSetValue(int value);

};


#endif

// Master Thesis - StoneFace v.1.0 beta C by Jorge Garcia Bueno
// Universidad Carlos III de Madrid + University of Glasgow 2009
// All rights reserved by the author
// -------------------------------------------------------------
// mixeraut.cpp - Mixing warped images in automatic mode

#include "mixeraut.h"

MixerAutomatic::MixerAutomatic(StoneFaceWidget *mainw, AutomaticWidget *qwx)
{
    widget = mainw;
    qw = qwx;



}

int MixerAutomatic::setFilesList(QStringList l)
```

```
{
    files = &l;
}
int MixerAutomatic::SearchFiles()
{
    files = new QStringList(QFileDialog::getOpenFileNames(
                         0,
                         "Select one or more files to open",
                         "./",
                         "Images (*.png *.xpm *.jpg *.bmp *.gif)"));
return files->size();
}

int MixerAutomatic::Start(QString outputname,int imgw,int imgh,bool grayscale)
{

    // Ensure there is a list of files...
    if (files == NULL) return 0;

    //QStringList list = files;
    QStringList::Iterator it = files->begin();
    QString s;

    QImage *avgPic = new QImage(180,180,QImage::Format_RGB32);
    QList<qreal> avgMesh;

    unsigned int avgPicArray[3*180][180];
    int r,g,b;
    QColor *co;
    int k = 0,nTotal=files->size(), counter = 0;

    if (nTotal == 0) return (0);


    // Create the list and the array with 0 values
    for (int i = 0; i < 24; i++) avgMesh.append(0);

    for (int i=0;i<180;i++)
      for (int j=0;j<180*3;j++)
        avgPicArray[j][i]=0;


    //Necesary to check that all the images have the same size ...

    int hlp=0,totalOrig=files->size();
    // Start iterations
    while( it != files->end() )
    {
        counter++;
  //      if (grayscale == true)
  //          widget = new StoneFaceWidget();
  //      else
            widget = new StoneFaceWidget();

        // We will reject the picture if any of the elements of the face is found
        //fprintf(stderr, "\nTarget: %s\n", (char*)s.toLatin1().data());
        if (widget->loadPicture(*it,(char*)"tinyposaut.dat",(char*)"tinyposaut2.dat",(char*)"tinytriangles
aut.dat",true,true) == 0 &&  widget->updatePicture() == 0)

        {

            s = *it;
            hlp++;

            // emision of a signal with the info to be displayed
            emit SetStatus(*it,counter,totalOrig,widget->currentImage().depth(),
                            widget->currentImage().numColors(), widget->currentImage().numBytes(),
                            widget->currentImage().width(),widget->currentImage().height(),
                            QImage(*it),widget->currentImage());
            widget->downloadAddMesh(&avgMesh);
            //printf("value element 18 avgMesh %f %f\n",avgMesh.value(34),avgMesh.value(35));
            for (int i=0;i<180;i++)
            {
              for (int j=0;j<180*3;j=j+3)
              {
                co = new QColor(widget->currentImage().pixel(k++,i));

                co->getRgb(&r,&g,&b);
                avgPicArray[j][i]   += r;
                avgPicArray[j+1][i] += g;
                avgPicArray[j+2][i] += b;

              }k=0;
            }


            s.append("-morphed.jpg");
            //widget->savePicture(s);
            printf("State: Accepted!\n");
            ++it;
        }

        else
        {
            s = *it;
            nTotal--;
            printf("State: Rejected!\n");
            emit SetStatus(*it,counter,totalOrig,widget->currentImage().depth(),
                            widget->currentImage().numColors(), widget->currentImage().numBytes(),
                            widget->currentImage().width(),widget->currentImage().height(),
                            widget->currentImage(),widget->currentImage());

            //delete widget;
            ++it;
        }

    PBSetValue((int) (counter/nTotal));
    }

    k=0;

    //Average of the image, the vector and set the pixel
    for (int i=0;i<180;i++)
    {
      for (int j=0;j<180*3;j=j+3)
        {

            avgPicArray[j][i]   = (unsigned char)(avgPicArray[j+0][i]/nTotal);
            avgPicArray[j+1][i] = (unsigned char)(avgPicArray[j+1][i]/nTotal);
            avgPicArray[j+2][i] = (unsigned char)(avgPicArray[j+2][i]/nTotal);

            co = new QColor(avgPicArray[j][i],avgPicArray[j+1][i],avgPicArray[j+2][i]);
            avgPic->setPixel(k++,i,co->rgb());
        }k=0;
    }

    for (int i = 0;i < 24;i++)
        avgMesh.replace(i,avgMesh.value(i) / nTotal);

    fprintf(stderr, "\n\n# Number of faces Accepted: %d\n\n", nTotal);


// Save the average mesh in a file to be sent to the widget object
    FILE* f=fopen("avg.dat","w");

    if (!f) {
        fprintf(stderr,"Error saving the mesh file!\n");
        exit(1);
    }
    //Print the number of nodes first
    fprintf(f,"12\n");

    for (int i = 0;i < 24;i = i +2)
        fprintf(f,"%d %d o\n",(int)avgMesh.value(i),(int)avgMesh.value(i+1));

    fclose(f);


    if (grayscale== true)
    {
        int grayColor = 0;int rx,gx,bx;
        for (int i=0;i<180;i++)
        {
          for (int j=0;j<180;j++)
            {
                co = new QColor(avgPic->pixel(j,i));
                co->getRgb(&rx,&gx,&bx);
                grayColor = (rx + gx + bx )/3;
                co->setRgb(grayColor,grayColor,grayColor);
                avgPic->setPixel(i,j,co->rgb());
            }k=0;
        }
    }
        avgPic = new QImage(avgPic->scaled(190,285,Qt::IgnoreAspectRatio,Qt::SmoothTransformation) );
        avgPic->save(outputname);


// Now we need to deform back the picture to the avg mesh

        const QString avgee = QString(outputname) ;

        widget = new StoneFaceWidget();
        //printf("ruta: %s", outputname->);
        widget->loadPicture(outputname,(char*)"tinyposaut.dat",(char*)"avg.dat", (char*)"tinytrianglesaut.
dat",true,false);
        widget->updatePicture();
       // widget->savePicture("aamorphedAverage.jpg");
    if (imgw != 0 && imgh != 0)
        avgPic = new QImage(widget->currentImage().scaled(imgw,imgh,Qt::IgnoreAspectRatio,Qt::SmoothTransf
ormation) );
        avgPic->save(outputname);

return (1);
}

// Internal methods to update the status of the Automatic Widget

void MixerAutomatic::PBSetValue(int value)
{
    if (qw != NULL)
        qw->ProgressBarSetValue(value);
}


// Master Thesis - StoneFace v.1.0 beta C by Jorge Garcia Bueno
// Universidad Carlos III de Madrid + University of Glasgow 2009
// All rights reserved by the author
// -------------------------------------------------------------
// node.h - Header with the instructions for node.cpp

#ifndef NODE_H
#define NODE_H

#include <QGraphicsItem>
#include <QList>


class GraphWidget;
class QGraphicsSceneMouseEvent;
class Node : public QGraphicsItem
{
public:
    Node(GraphWidget *graphWidget, int nodeNumber, char color, bool helpNode);


    enum { Type = UserType + 1 };
    int type() const { return Type; }

    qreal posX();
    qreal posY();
    void calculateForces();
    bool advance();

    QRectF boundingRect() const;
    QPainterPath shape() const;
    void paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *widget);
    char getColor();
    void setColor(char);
    bool helpNode;
protected:
    QVariant itemChange(GraphicsItemChange change, const QVariant &value);

    void mousePressEvent(QGraphicsSceneMouseEvent *event);
    void mouseReleaseEvent(QGraphicsSceneMouseEvent *event);

private:

    QPointF newPos;
    GraphWidget *graph;
    int nNumber;
    char color;

};

#endif

// Master Thesis - StoneFace v.1.0 beta C by Jorge Garcia Bueno
// Universidad Carlos III de Madrid + University of Glasgow 2009
// All rights reserved by the author
// -------------------------------------------------------------
// node.cpp - Individual node to interact with others in meshes

#include <QGraphicsScene>
#include <QGraphicsSceneMouseEvent>
#include <QPainter>
#include <QStyleOption>


#include "node.h"
#include "graphwidget.h"

Node::Node(GraphWidget *graphWidget, int nodeNumber, char _color, bool _helpNode)
    : graph(graphWidget)
{
    //setCacheMode(DeviceCoordinateCache);
    color = _color;
    setFlag(ItemIsMovable);
    setZValue(1);
    nNumber = nodeNumber;
    helpNode= _helpNode;

}


void Node::calculateForces()
{

    if (helpNode==false) graph->nodeMoved(nNumber,newPos.x(),newPos.y());
    if (helpNode==false) newPos = pos();

}

bool Node::advance()
{
    if (newPos == pos())
        return false;

    if (helpNode==false) setPos(newPos);

    return true;
}
char Node::getColor()
{
    return color;
}
void Node::setColor(char _color)
{
    color = _color;
}
QRectF Node::boundingRect() const
{
    qreal adjust = 2;
    return QRectF(-10 - adjust, -10 - adjust,23 + adjust, 23 + adjust);

}

QPainterPath Node::shape() const
{
    QPainterPath path;
    path.addEllipse(-5, -5, 10, 10);

    return path;
}

void Node::paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *)
{
    if (option->state & QStyle::State_Sunken)
    {
        painter->setPen(QPen(Qt::darkRed, 3));
    }
    else
    {
        switch (color)
        {
            case 'b':    painter->setPen(QPen(QColor(67, 0,239), 3));break;
            case 'w':    painter->setPen(QPen(QColor(0,255,255), 3));break;
            case 'y':    painter->setPen(QPen(QColor(255,255,0), 3));break;
            case 'o':    painter->setPen(QPen(QColor(255,128,0), 3));break;
            case 'p':    painter->setPen(QPen(QColor(255,0,255), 3));break;
            case 'r':    painter->setPen(QPen(QColor(255,0,  0), 3));break;
            case 'v':    painter->setPen(QPen(QColor(181,0,255), 3));break;
            case 'g':    painter->setPen(QPen(QColor(80,255, 0), 3));break;
            case 'm':    painter->setPen(QPen(QColor(0,255,  0), 3));break;
            case 'k':    painter->setPen(QPen(QColor(255,255,255), 3));break;
            case 't':    painter->setPen(QPen(Qt::transparent));break;
        }

    painter->drawEllipse(-2, -2, 4, 4);
    }
}

QVariant Node::itemChange(GraphicsItemChange change, const QVariant &value)
{
    switch (change) {
    case ItemPositionHasChanged:

        if (helpNode==false)graph->itemMoved();
        break;
    default:
        break;
    };

    return QGraphicsItem::itemChange(change, value);
}

void Node::mousePressEvent(QGraphicsSceneMouseEvent *event)
{
    update();
    printf("node#%d position:(%d,%d)\n",nNumber,(int)posX(),(int)posY() );
    if (helpNode==false) graph->nodeClicked(nNumber);
    if (helpNode==false) QGraphicsItem::mousePressEvent(event);
}
qreal Node::posX()
    {return newPos.x();}
qreal Node::posY()
    {return newPos.y();}


void Node::mouseReleaseEvent(QGraphicsSceneMouseEvent *event)
{

    update();

    QGraphicsItem::mouseReleaseEvent(event);
}

// Master Thesis - StoneFace v.1.0 beta C by Jorge Garcia Bueno
// Universidad Carlos III de Madrid + University of Glasgow 2009
// All rights reserved by the author
// ---------------------------------------------------------------
// nodeaut.h - Header with the instructions for nodeaut.cpp

#ifndef NODEAUTOM_H
#define NODEAUTOM_H

#include <QGraphicsItem>
#include <QList>
#include "stonefacewidget.h"

class StoneFaceWidget;
class QGraphicsSceneMouseEvent;

class NodeAutomatic : public QGraphicsItem
{
public:
    NodeAutomatic(StoneFaceWidget *graphWidget, int nodeNumber, char color);


    enum { Type = UserType + 1 };
    int type() const { return Type; }

    qreal posX();
    qreal posY();
    void calculateForces();
    bool advance();

    QRectF boundingRect() const;
    QPainterPath shape() const;
    void paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *widget);
    char getColor();

protected:
    QVariant itemChange(GraphicsItemChange change, const QVariant &value);

    void mousePressEvent(QGraphicsSceneMouseEvent *event);
    void mouseReleaseEvent(QGraphicsSceneMouseEvent *event);
```

```
private:

    QPointF newPos;
    StoneFaceWidget *graph;
    int nNumber;
    char color;

};

#endif

// Master Thesis - StoneFace v.1.0 beta C by Jorge Garcia Bueno
// Universidad Carlos III de Madrid + University of Glasgow 2009
// All rights reserved by the author
// -------------------------------------------------------------
// nodeaut.cpp - Class to control automatic nodes (less complex than manual nodes)

#include <QGraphicsScene>
#include <QGraphicsSceneMouseEvent>
#include <QPainter>
#include <QStyleOption>


#include "nodeaut.h"


NodeAutomatic::NodeAutomatic(StoneFaceWidget *graphWidget, int nodeNumber, char _color)
    : graph(graphWidget)
{
    color = _color;
    setFlag(ItemIsMovable);
    setZValue(1);
    nNumber = nodeNumber;
}


void NodeAutomatic::calculateForces()
{
    //if (!scene() || scene()->mouseGrabberItem() == this) {              //-------Currently just returns
new position as whatever position its at.
    graph->nodeMoved(nNumber,newPos.x(),newPos.y());
        newPos = pos();
    //}

}

bool NodeAutomatic::advance()
{
    if (newPos == pos())
        return false;

    setPos(newPos);

    return true;
}
char NodeAutomatic::getColor()
{
    return color;
}

QRectF NodeAutomatic::boundingRect() const
{
    qreal adjust = 2;
    return QRectF(-10 - adjust, -10 - adjust,
                  23 + adjust, 23 + adjust);

}

QPainterPath NodeAutomatic::shape() const
```

```
{
    QPainterPath path;
    path.addEllipse(-5, -5, 10, 10);

    return path;
}

void NodeAutomatic::paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *)
{
    if (option->state & QStyle::State_Sunken)
    {
        painter->setPen(QPen(Qt::red, 4));
    }
    else
    {
        switch (color)
        {
            case 'b':    painter->setPen(QPen(QColor(67,0,239), 4));break;
            case 'w':    painter->setPen(QPen(QColor(0,255,255), 4));break;
            case 'y':    painter->setPen(QPen(QColor(255,255,0), 4));break;
            case 'o':    painter->setPen(QPen(QColor(255,128,0), 4));break;
            case 'p':    painter->setPen(QPen(QColor(255,0,255), 4));break;
            case 'r':    painter->setPen(QPen(QColor(255,0,0), 4));break;
            case 'v':    painter->setPen(QPen(QColor(181,0,255), 4));break;
            case 'g':    painter->setPen(QPen(QColor(80,255,0), 4));break;
            case 't':    painter->setPen(QPen(Qt::transparent));break;
        }

    painter->drawEllipse(-2, -2, 4, 4);
    }
}

QVariant NodeAutomatic::itemChange(GraphicsItemChange change, const QVariant &value)
{
    switch (change) {
    case ItemPositionHasChanged:
        graph->itemMoved();

        break;
    default:
        break;
    };

    return QGraphicsItem::itemChange(change, value);
}

void NodeAutomatic::mousePressEvent(QGraphicsSceneMouseEvent *event)
{
    printf("node#%d position:(%d,%d)\n",nNumber,(int)posX(),(int)posY() );
    QGraphicsItem::mousePressEvent(event);
}
qreal NodeAutomatic::posX()
    {return newPos.x();}
qreal NodeAutomatic::posY()
    {return newPos.y();}


void NodeAutomatic::mouseReleaseEvent(QGraphicsSceneMouseEvent *event)
{

    update();

    QGraphicsItem::mouseReleaseEvent(event);
}

// Master Thesis - StoneFace v.1.0 beta C by Jorge Garcia Bueno
// Universidad Carlos III de Madrid + University of Glasgow 2009
// All rights reserved by the author
// ------------------------------------------------------------
// searchwidget.h - Header with the instructions for searchwidget.cpp
```

```
#ifndef SEARCHFACEWIDGET_H
#define SEARCHFACEWIDGET_H

#include <QtGui>
#include <QPixmap>
#include <QGraphicsView>
#include <QListView>
#include <QDialog>
#include <QPushButton>
#include <QDesktopWidget>
#include "mainwindow.h"
#include "eigenfaces.h"

class EigenFaces;

class SearchWidget : public QWidget
{
    Q_OBJECT


private:
    // MainWindow handle
    QMainWindow *mainWin;

    void StartConnections();

    // Eigenfaces objects
    EigenFaces *eigen;

    // Inteface objects

    QPushButton *buttonSelectDB;
    QPushButton *buttonSelectFace;
    QPushButton *buttonStartSearch;

    QLabel *infoDB;
    QLabel *infoFace;
    QLabel *instructionsLbl;
    QLabel *instructionsLbl2;
    QLabel *mixInfo;
    QLabel *face1;
    QProgressBar *pb;
    QString *pathImageSelected;
    QGridLayout *lLayout;

    void StartInterface();
    void MoveCenterScreen();
public:
    SearchWidget(QMainWindow *parent);

private slots:

void SelectDB();
void SelectFace();
void StartSearch();

public slots:
void GetInfoDB(QString , int , int , int );
void GetNearest(QString);
void GetProgress(int);
void InfoNearest(QString);
};
#endif

// Master Thesis - StoneFace v.1.0 beta C by Jorge Garcia Bueno
// Universidad Carlos III de Madrid + University of Glasgow 2009
// All rights reserved by the author
// -------------------------------------------------------------
// searchwidget.cpp - Interface to control the PCA

#include "searchwidget.h"

SearchWidget::SearchWidget(QMainWindow *parent)
{
    // Remember the handle of the parent for the returning function
    mainWin = parent;

    // Start the interactive objects
    StartInterface();
    MoveCenterScreen();
    StartConnections();

}

void SearchWidget::MoveCenterScreen()
{

    int screenWidth, width;
    int screenHeight, height;


    QDesktopWidget *desktop = new QDesktopWidget();
    QSize windowSize;
    screenWidth = desktop->width(); // get width of screen
    screenHeight = desktop->height(); // get height of screen

    windowSize = size(); // size of our application window
    width = windowSize.width();
    height = windowSize.height();

    move((screenWidth-width)/2,(screenHeight - height)/2);
}

void SearchWidget::StartConnections()
{
    connect(buttonSelectDB, SIGNAL(clicked()), this, SLOT( SelectDB() ) );
    connect(buttonSelectFace, SIGNAL(clicked()), this, SLOT(SelectFace()));
    connect(buttonStartSearch, SIGNAL(clicked()), this, SLOT(StartSearch()));
}

void SearchWidget::SelectDB()
{

    QString fileName = QFileDialog::getOpenFileName(this, tr("Select the face database file"),
                                                    "./", tr("Face databases (*.xml)"));

    if (fileName.isEmpty() == false)
    {
        eigen = new EigenFaces(this);
        connect (eigen, SIGNAL(InfoDB(QString , int , int , int )),
                this, SLOT(GetInfoDB(QString , int , int , int )));

        connect (eigen, SIGNAL(NearestImage(QString)),
                this, SLOT(GetNearest(QString)));

        connect (eigen, SIGNAL(InfoNearest(QString)),
                this, SLOT(InfoNearest(QString)));

        connect (eigen, SIGNAL(Progress(int)),
                this, SLOT(GetProgress(int)));

        eigen->SetDB(fileName.toLatin1().data());
    }
}
void SearchWidget::SelectFace()
{
    QString fileName = QFileDialog::getOpenFileName(this, tr("Select the face to recognise"),
                                                    "./", tr("Images (*.png *.xpm *.jpg *.bmp)"));
    if (fileName.isEmpty() == false)
    {
        pathImageSelected = new QString(fileName);

        QImage img(fileName);

        QString mixInfoL("");
        mixInfoL += "Information:\nPath: ..." + pathImageSelected->right(20) + "\n";
        mixInfoL += "Depth: " + QString::number(img.depth()) + "\n";
        mixInfoL += "Number of Colors: " + QString::number(img.numColors()) + " colors\n";
        mixInfoL += "Number of bytes: " + QString::number(img.numBytes()) + " bytes\n";
        mixInfoL += "Size: " + QString::number(img.width()) + " x " + QString::number(img.height() );
        mixInfo->setText(mixInfoL);
    }

}
void SearchWidget::StartSearch()
{
    if (pathImageSelected->isEmpty() == false)
    {
        eigen->Recognise(pathImageSelected->toLatin1().data(), true);

    }
}


void SearchWidget::StartInterface()
{
    instructionsLbl = new QLabel("Step #1 - Please, select the face database file you want to use", this);
    instructionsLbl2 = new QLabel("Step #2 - Select the face you need to recognise", this);

    QString infoDBL("");
    infoDBL += "Information of the database:\n";
    infoDBL += "Date: \n";
    infoDBL += "Number of faces: \n";
    infoDBL += "Size of faces: \n";

    infoDB = new QLabel(infoDBL);

    QString infoFaceL("");
    infoFaceL += "Information about the closest face:\nPath: ...\n";
    infoFaceL += "Name: \n";
    infoFaceL += "Age: \n";
    infoFaceL += "Gender: \n";
    infoFaceL += "Nacionality: \n";
    infoFaceL += "Proffession:";
    infoFace = new QLabel(infoFaceL);

    buttonSelectDB = new QPushButton("&Select DB file ...", this);
    buttonSelectFace = new QPushButton("&Select face ...", this);
    buttonStartSearch = new QPushButton("&Recognise this face ...", this);

    face1 = new QLabel("Closes face",this);
    face1->setPixmap(QPixmap("./images/user.png"));

    QString mixInfoL("");
    mixInfoL += "Information:\nPath: ...\n";
    mixInfoL += "Depth: \n";
    mixInfoL += "Number of Colors: \n";
    mixInfoL += "Number of bytes: \n";
    mixInfoL += "Size: ";
    mixInfo = new QLabel(mixInfoL);

    pb = new QProgressBar(this);
    pb->setValue(0);
    // Creation of layout to define the positions of the objects
    lLayout = new QGridLayout;
    lLayout->addWidget(instructionsLbl,0,0,1,2);
    lLayout->addWidget(buttonSelectDB,1,0,1,1);
    lLayout->addWidget(infoDB,1,1,1,1);

    // ------------------------------------------

    lLayout->addWidget(instructionsLbl2,3,0,1,2);
    lLayout->addWidget(buttonSelectFace,4,0,2,1);
    lLayout->addWidget(mixInfo,4,1,2,1);

    // ------------------------------------------

    lLayout->addWidget(infoFace,6,0,1,1);
    lLayout->addWidget(face1,6,1,2,1);
    lLayout->addWidget(buttonStartSearch,7,0,1,1);
    // ------------------------------------------

    lLayout->addWidget(pb,8,0,1,2);

    setLayout(lLayout);
}

void SearchWidget::GetInfoDB(QString dateS, int numFaces, int widthFaces, int heightFaces)
{

    QString infoDBL("");
    infoDBL += "Information of the database:\n";
    infoDBL += "Date: " + dateS + "\n";
    infoDBL += "Number of faces: " + QString::number(numFaces) + "\n";
    infoDBL += "Size of faces: "+ QString::number(widthFaces) + " x " + QString::number(heightFaces);

    infoDB->setText(infoDBL);

}
void SearchWidget::GetNearest(QString pathNearest)
{
    printf("path nearest: %s\n",pathNearest.toLatin1().data());
    face1->setPixmap(QPixmap(QPixmap(pathNearest).scaled(120,180)));

}
void SearchWidget::InfoNearest(QString infoSubject)
{

    infoFace->setText(infoSubject);

}

void SearchWidget::GetProgress(int p)
{
    pb->setValue(p);
}

// Master Thesis - StoneFace v.1.0 beta C by Jorge Garcia Bueno
// Universidad Carlos III de Madrid + University of Glasgow 2009
// All rights reserved by the author
// --------------------------------------------------------------
// stonefacewidget.h - Header with the instructions for stonefacewidget.cpp

#ifndef STONEWIDGET_H
#define STONEWIDGET_H

#include <QtGui/QGraphicsView>
#include "nodeaut.h"
#include "detector.h"


class NodeAutomatic;

class StoneFaceWidget : public QGraphicsView
{
    Q_OBJECT
```

```
public:
    StoneFaceWidget();

    void itemMoved();
    QImage currentImage();
    void nodeMoved(int nodeNumber, qreal newX,qreal newY);
    void savePicture(const QString &fileName);
    int loadPicture(const QString &fileName, char* origMesh, char* destMesh,char* triang,bool tiny, bool a
utodetection);
    int updatePicture();
    void downloadAddMesh(QList<qreal>*);

protected:

    void timerEvent(QTimerEvent *event);

    int aNodes[38][2];
    int aNodes2[38][2];
    int aTri[70][3];
    int nTri;
    bool inside, flag;

    double A[3][3];
    double in[3][3], alpha, beta, gamma;
    int Apx,Apy,Bpx,Bpy,Cpx,Cpy,Xp,Yp;
    QRgb valor;
    double detInv;//n is the determinant of A
private:
    QString *widgetFileName;
    QPixmap *fa1,*fa2;
    long lineLength(int A1, int B1, int A2, int B2);
    double areaOfTriangle(int Ax, int Ay, int Bx, int By, int Cx, int Cy);
    int detectElements(bool stretch);
    int detectElementsTiny(bool strech);
    int timerId;
    bool insideTriangle(int Ax, int Ay, int Bx, int By, int Cx, int Cy, int Px, int Py);
    QGraphicsPixmapItem *face1,*face2;
    QImage *imf1,*imf2;
    QGraphicsScene *superScene;

signals:
    void zoneScanned(QString);
};

#endif

// Master Thesis - StoneFace v.1.0 beta C by Jorge Garcia Bueno
// Universidad Carlos III de Madrid + University of Glasgow 2009
// All rights reserved by the author
// ---------------------------------------------------------------
// stonefacewidget.cpp - Interface to control the automatic generated warp images

#include "stonefacewidget.h"


#include <QDebug>
#include <QGraphicsScene>

#include <QtGui>
#include <QPixmap>
#include <QGraphicsView>
#include <math.h>
#include <QPushButton>
StoneFaceWidget::StoneFaceWidget()
    : timerId(0)
{

    QGraphicsScene *scene = new QGraphicsScene(this);
    scene = new QGraphicsScene(this);
    superScene = scene;
    scene->setItemIndexMethod(QGraphicsScene::NoIndex);
    scene->setSceneRect(0,0, 480, 300);
    setScene(scene);

}

void StoneFaceWidget::itemMoved()
{
    if (!timerId)
        timerId = startTimer(1 / 25);
}

void StoneFaceWidget::nodeMoved(int nodeNumber, qreal newX,qreal newY)
{
    // Refresh the picture with the new image

    aNodes[nodeNumber][0] = (int)newX;
    aNodes[nodeNumber][1] = (int)newY;

}



void StoneFaceWidget::timerEvent(QTimerEvent *event)
{
    Q_UNUSED(event);

    QList<NodeAutomatic *> nodes;
    foreach (QGraphicsItem *item, scene()->items()) {
        if (NodeAutomatic *node = qgraphicsitem_cast<NodeAutomatic *>(item))
            nodes << node;
    }

    foreach (NodeAutomatic *node, nodes)
        node->calculateForces();

    bool itemsMoved = false;
    foreach (NodeAutomatic *node, nodes) {
        if (node->advance())
            itemsMoved = true;
    }

    if (!itemsMoved) {
        killTimer(timerId);
        timerId = 0;
    }
}

QImage StoneFaceWidget::currentImage() {return *imf2;}

int StoneFaceWidget::loadPicture(const QString &fileName, char* origMesh, char* destMesh, char* triang,boo
l tiny, bool autodetection)
{
//--get background image----------------------------------------------
  printf("loading...\n");
  widgetFileName = new QString(fileName.toLatin1().data());
  fa1 = new QPixmap(fileName);
  fa2 = new QPixmap(fileName);

  imf1 = new QImage(fa1->toImage());
  imf2 = new QImage(fa2->toImage());

// Resize the pictures
  //*fa1 = fa1->scaled(195,285,Qt::IgnoreAspectRatio,Qt::SmoothTransformation);
  //*imf1 = imf1->scaled(195,285,Qt::IgnoreAspectRatio,Qt::SmoothTransformation);

  face1 = new QGraphicsPixmapItem(*fa1);
  face2 = new QGraphicsPixmapItem(*fa2);

  superScene->addItem(face1);
  //superScene->addItem(face2);
  face1->setPos(0,0);
  //face2->setPos(240,0);

//--get position of points from data-----------------------------------


// by default we position all the nodes in the final destination. After the recognition the positions wil
l change
    FILE* f=fopen(origMesh,"r");
    if (!f) {
        fprintf(stderr,"Original mesh: %s not there!\n",origMesh);
        exit(1);
    }

    int nNodes, x, y, z;
    char color;
    z = fscanf(f,"%d\n",&nNodes);
    int nn = nNodes;

    NodeAutomatic** node=new NodeAutomatic*[nn];

    for(int i=0; i<nn; i++)
    {
        z = fscanf(f,"%d %d %c\n",&x,&y,&color);
        node[i]= new NodeAutomatic(this,i,color);
    //if (i != 0 && i != 1 && i != 2 && i != 17 && i != 18 && i != 27 && i != 28 && i != 29 && i != 30 &&
i != 31 && i != 32 && i != 33 )
        superScene->addItem(node[i]);
        node[i]->setPos(x,y);
        aNodes[i][0]=x;
        aNodes[i][1]=y;

        node[i]->calculateForces();
    }


    fclose(f);


// Read the points from the file2 -> final points
    FILE* f3=fopen(destMesh,"r");
    if (!f3) {
        fprintf(stderr,"Destination mesh: %s not there!\n",destMesh);
        exit(-1);
        }

    int nNodes2;
    char c;
    z = fscanf(f3,"%d\n",&nNodes2);

    for (int k=0;k<nNodes2;k++)
    {
        z = fscanf(f3,"%d %d %c\n",&aNodes2[k][0],&aNodes2[k][1], &c);
    }

    fclose(f3);
/*
*/
// Read the points to know the triangles done
    FILE* f4=fopen(triang,"r");
    if (!f4){
        fprintf(stderr,"Triangles data: %s not there!\n", triang);
        exit(-1);
        }


    z = fscanf(f4,"%d\n",&nTri);

    for (int k=0;k<nTri;k++)
    {
        z = fscanf(f4,"%d %d %d\n",&aTri[k][0],&aTri[k][1],&aTri[k][2]);
    }

    fclose(f4);


    // Refresh the points
        QList<NodeAutomatic *> nodes;
        foreach (QGraphicsItem *item, scene()->items())
            if (NodeAutomatic *node = qgraphicsitem_cast<NodeAutomatic *>(item))
                nodes << node;

        foreach (NodeAutomatic *node, nodes) node->calculateForces();

    if (autodetection == true)
    {
        if (tiny == true)
        {

            if (detectElementsTiny(false) == 0)
            {

            // Refresh the points
                QList<NodeAutomatic *> nodes;
                foreach (QGraphicsItem *item, scene()->items())
                    if (NodeAutomatic *node = qgraphicsitem_cast<NodeAutomatic *>(item))
                        nodes << node;

                foreach (NodeAutomatic *node, nodes) node->calculateForces();

            }
            else
            {
                return 1;
            }
        }
        else
        {

            if (detectElements(false) == 0)
            {

            // Refresh the points
                QList<NodeAutomatic *> nodes;
                foreach (QGraphicsItem *item, scene()->items())
                    if (NodeAutomatic *node = qgraphicsitem_cast<NodeAutomatic *>(item))
                        nodes << node;

                foreach (NodeAutomatic *node, nodes) node->calculateForces();

            }
            else
            {
                return 1;
            }
        }
    }
return 0;
}
void StoneFaceWidget::savePicture(const QString &fileName)
{

  imf2->save(fileName);


}

void StoneFaceWidget::downloadAddMesh(QList<qreal> *list)
{
    int i=0;
    QList<NodeAutomatic *> nodes;
    foreach (QGraphicsItem *item, scene()->items()) {
        if (NodeAutomatic *node = qgraphicsitem_cast<NodeAutomatic *>(item))
            nodes << node;

    }
    printf("list size: %d\n",list->size() );
// the lenght of this list must be the same as 2*the number of nodes in the mesh
    if (list->size() == 24)
    {

        foreach (NodeAutomatic *node, nodes)
        {

            list->replace(i,list->value(i) + node->posX());
            list->replace(i+1,list->value(i+1) + node->posY());
            printf("nodeX %f , nodeY %f\n", node->posX(), node->posY());
            i=i+2;
        }
    }
}



int StoneFaceWidget::updatePicture()
{

//Pre selection of the size of the head 180 x 180
int mTriangles[ imf1->width()][imf1->height()];

bool flag = false;
for (int j = 0; j < imf1->height();j++){
for (int i = 0; i < imf1->width();i++){
    // For each triangle in the image...
    for (int k = 0; k < nTri; k ++)
        {

            inside = insideTriangle(aNodes2[ aTri[k][0]][0], aNodes2[ aTri[k][0]][1], aNodes2[ aTri[k][1]]
[0], aNodes2[ aTri[k][1]][1], aNodes2[ aTri[k][2]][0], aNodes2[ aTri[k][2]][1],i,j);
            if (inside == true)
            {
                mTriangles[i][j] = k;

                flag = true;
            }
        }
    if (flag == false) {printf("Point out of matrix ?? point X(%d,%d)\n",i, j);return(-1);}
    flag = false;
}
}

// Now, creation of the matrix and compute the inverse...
for (int j = 0; j < imf1->height();j++){
for (int i = 0; i < imf1->width();i++){


    //Point A of the triangle in which the point (i,j) is located
    A[0][0] = aNodes2 [aTri[ mTriangles[i][j] ][0]][0];
    A[1][0] = aNodes2 [aTri[ mTriangles[i][j] ][0]][1];
    A[2][0] = 1;

    //Point B of the triangle in which the point (i,j) is located
    A[0][1] = aNodes2 [aTri[ mTriangles[i][j] ][1]][0];
    A[1][1] = aNodes2 [aTri[ mTriangles[i][j] ][1]][1];
    A[2][1] = 1;

    //Point C of the triangle in which the point (i,j) is located
    A[0][2] = aNodes2 [aTri[ mTriangles[i][j] ][2]][0];
    A[1][2] = aNodes2 [aTri[ mTriangles[i][j] ][2]][1];
    A[2][2] = 1;

//  printf("A[0][0] = [%d] A[0][1] = [%d] A[0][2] = [%d]\n", A[0][0],A[0][1],A[0][2]);
//  printf("A[1][0] = [%d] A[1][1] = [%d] A[1][2] = [%d]\n", A[1][0],A[1][1],A[1][2]);
//  printf("A[2][0] = [%d] A[2][1] = [%d] A[2][2] = [%d]\n\n", A[2][0],A[2][1],A[2][2]);
//Inverse....

detInv = (  A[0][0] * (A[1][1] * A[2][2] - A[1][2] * A[2][1]) -
        A[0][1] * (A[1][0] * A[2][2] - A[1][2] * A[2][0]) +
        A[0][2] * (A[1][0] * A[2][1] - A[1][1] * A[2][0]));
    if( detInv == 0.0 ) {printf("Determinant null! %d %d %f\n",i,j, A[1][1]);break;}

    detInv = (double)(1.0 / detInv);
    in[0][0] =  detInv * (A[1][1] * A[2][2] - A[1][2] * A[2][1]);
    in[0][1] = -detInv * (A[0][1] * A[2][2] - A[0][2] * A[2][1]);
    in[0][2] =  detInv * (A[0][1] * A[1][2] - A[0][2] * A[1][1]);

    in[1][0] = -detInv * (A[1][0] * A[2][2] - A[1][2] * A[2][0]);
    in[1][1] =  detInv * (A[0][0] * A[2][2] - A[0][2] * A[2][0]);
    in[1][2] = -detInv * (A[0][0] * A[1][2] - A[0][2] * A[1][0]);

    in[2][0] =  detInv * (A[1][0] * A[2][1] - A[1][1] * A[2][0]);
    in[2][1] = -detInv * (A[0][0] * A[2][1] - A[0][1] * A[2][0]);
    in[2][2] =  detInv * (A[0][0] * A[1][1] - A[0][1] * A[1][0]);


//Then, once inverted, multiply by the values of the position of X(i,j)

    alpha = in[0][0]*i + in[0][1]*j + in[0][2];
    beta =  in[1][0]*i + in[1][1]*j + in[1][2];
    gamma = in[2][0]*i + in[2][1]*j + in[2][2];

//Now

    Apx = aNodes [aTri[ mTriangles[i][j] ][0]][0];           //Point Ap of the triangle (original triangl
e)
    Apy = aNodes [aTri[ mTriangles[i][j] ][0]][1];

    Bpx = aNodes [aTri[ mTriangles[i][j] ][1]][0];           //Point Bp of the triangle (original triangl
e)
    Bpy = aNodes [aTri[ mTriangles[i][j] ][1]][1];

    Cpx = aNodes [aTri[ mTriangles[i][j] ][2]][0];           //Point Cp of the triangle (original triangl
e)
    Cpy = aNodes [aTri[ mTriangles[i][j] ][2]][1];


    Xp = alpha * Apx + beta * Bpx + gamma * Cpx;
    Yp = alpha * Apy + beta * Bpy + gamma * Cpy;


    valor = imf1->pixel(Xp,Yp);

    QColor color(valor);
    imf2->setPixel(i,j,color.rgb());

}

}

    //superScene->update(0,0,superScene->width(),superScene->height());
    QPixmap *k = new QPixmap(imf1->width(),imf1->height());
    face1->setPixmap(k->fromImage(*imf2));
    return (0);
}

int StoneFaceWidget::detectElements(bool stretch)
{

    IplImage *ImageFace,*subImageFace, *subImageEyeR, *subImageEyeL, *subImageNose, *subImageMouth, *subIm
```

```
ageEyes;
    Detector *dFace, *dEyeR, *dEyeL, *dMouth, *dNose, *dEyes;
    int px,py,imHeight,imWidth;
    int h = 0;

    // Variables to obtain the boundary conditions

    int midYeyeL, midYeyeR; // Contains the medium height both eyes to determine the ears
    int faceX,faceY,faceW,faceH;
    int eyesX,eyesY,eyesW,eyesH;
    int mouthY,minYeye,maxXeyeL;
    int minXeyeR,avgXeyeR;
    int nodei = 0; // for list moving
    int extremeEyeR,extremeEyeL,avgXeyeL;
    // Creation of a list with the nodes to be moved

    QList<NodeAutomatic *> nodes;
    foreach (QGraphicsItem *item, scene()->items()) {
        if (NodeAutomatic *node = qgraphicsitem_cast<NodeAutomatic *>(item))
            nodes << node;
    }


    // ####################################################################################
    // -------------------- First: Face recognition  -------------------------------------
    // ####################################################################################


    // Extraction of the face of the person...
    ImageFace = cvCreateImage(cvSize(imf1->width(), imf1->height()), IPL_DEPTH_8U, 3);

    for (int i=0;i < imf1->width() * 3 * imf1->height(); i++)
        ((char*)ImageFace->imageData)[i] = *imf1->bits()+i;

    // Creation of the detector and obtaining of the coordinates for the face
    dFace =  new Detector(ImageFace);
    dFace->DetectAndDraw('F',&px,&py,&imHeight,&imWidth);

    // Emision of the signal detecting face ...
    emit zoneScanned(QString("face"));

    // Now we just want the face scaled with the coordinates changed and work with result

    QImage result = imf1->copy(px,py,imHeight,imWidth);
    result = result.scaled(180,180,Qt::IgnoreAspectRatio,Qt::SmoothTransformation);
    imf1 = new QImage(result);
    imf2 = new QImage(result);

    QPixmap *k = new QPixmap(result.width(),result.height());
    face1->setPixmap(k->fromImage(result));

    subImageFace = cvCreateImage(cvSize(result.width(), result.height()), IPL_DEPTH_8U, 3);

    for (int i=0;i < result.width() * 3 * result.height(); i++)
        ((char*)subImageFace->imageData)[i] = *result.bits()+i;

    // Extract the face picture in standard size
    QString imagePath(*widgetFileName);
    imagePath.insert(imagePath.size()-4,QString("_std"));
    fprintf(stderr,"path: %s\n",imagePath.toLatin1().data());
    //cvSaveImage(imagePath.toLatin1().data(),subImageFace);

    // Update the coordinates
    px = 0;py = 0;imHeight=result.height();imWidth=result.width();


    superScene->addRect((qreal)px,(qreal)py,(qreal)imWidth,(qreal)imHeight,  *(new QPen( * (new QColor(0,
0,0)))));
    fprintf(stderr,"# Face #\tx: %d\ty: %d\theight: %d\twidth: %d\n",px,py,imHeight,imWidth);
    // Fill the face boundary variables. Throught the face boundaries the rest of the elements will be pro
vided
    faceX = px; faceY = py; faceW = imWidth; faceH = imHeight;

    // Check there is a face recogniced to continue
    if (px == 0 && py == 0 && imWidth == 0 && imHeight == 0) return 1;
    // Move the nodes where we think they should be...

    // ####################################################################################
    // --------            Second: Both eyes recognition               -------------------
    // ####################################################################################
  printf("eyes...\n");

    subImageEyes = cvCreateImage(cvSize(result.width(), result.height()), IPL_DEPTH_8U, 3);

    // Clean the image
    for (int i = 0; i < subImageEyes->width*subImageEyes->height*3; i++) // over w*h*3 (channels)
            ((char*)subImageEyes->imageData)[i] = 255;

    // Copy the information between both image formats. Only the face region is needed to locate the eye s
quares
    h=faceX;
    for (int i = faceY ; i < faceY + faceH*3/4;i++)
    {
        for (int j = 3* faceX; j <  (faceX + faceW)*3; j+=3)
        {

            ((char*)subImageEyes->imageData)[i*result.width()*3 + j + 0] = qBlue(result.pixel(h,
i));    // blue channel
            ((char*)subImageEyes->imageData)[i*result.width()*3 + j + 1] = qGreen(result.pixel(h,i));  //
green channel
            ((char*)subImageEyes->imageData)[i*result.width()*3 + j + 2] = qRed(result.pixel(h,i));   // r
ed channel
            h++;
        }h=faceX;
    }


    // Save the image (debugging)
    //cvSaveImage("eyes.jpg",subImageEyes);

    //Creation of the detector object and obtention of the coordinates
    dEyes = new Detector(subImageEyes);
    dEyes->DetectAndDraw('E',&px,&py,&imHeight,&imWidth);

    // Emision of the signal detecting eyes ...
    emit zoneScanned(QString("eyes"));

    superScene->addRect((qreal)px,(qreal)py,(qreal)imWidth,(qreal)imHeight,  *(new QPen( * (new QColor(25
5,0,255)))));
    fprintf(stderr,"# Eyes #\tx: %d\ty: %d\theight: %d\twidth: %d\n",px,py,imHeight,imWidth);

    // Fill the eyes boundary variables. Throught the face boundaries the eyes will be provided
    eyesX = px; eyesY = py; eyesW = imWidth; eyesH = imHeight;

    // Check there are eyes
    if (px < 1 || py < 1 || imWidth < 1 || imHeight < 1) return 1;




    // ####################################################################################
    // -------- Third: Left eye recognition inside left half eyes region  ------------------
    // ####################################################################################


    subImageEyeL = cvCreateImage(cvSize(result.width(), result.height()), IPL_DEPTH_8U, 3);

    // Clean the image
    for (int i = 0; i < subImageEyeL->width*subImageEyeL->height*3; i++) // over w*h*3 (channels)
```

```
            ((char*)subImageEyeL->imageData)[i] = 255;

    // Copy the information between both image formats. Only the face region is needed to locate the eye s
quares
    h=eyesX;
    for (int i = eyesY ; i < eyesY + eyesH;i++)
    {
        for (int j = 3* eyesX; j <  (eyesX + eyesW/2)*3; j+=3)
        {

            ((char*)subImageEyeL->imageData)[i*result.width()*3 + j + 0] = qBlue(result.pixel(h,
i));    // blue channel
            ((char*)subImageEyeL->imageData)[i*result.width()*3 + j + 1] = qGreen(result.pixel(h,i));  //
green channel
            ((char*)subImageEyeL->imageData)[i*result.width()*3 + j + 2] = qRed(result.pixel(h,i));   // r
ed channel
            h++;
        }h=eyesX;
    }


    // Save the image (debugging)
    //cvSaveImage("left.jpg",subImageEyeL);

    //Creation of the detector object and obtention of the coordinates
    dEyeL = new Detector(subImageEyeL);
    dEyeL->DetectAndDraw('L',&px,&py,&imHeight,&imWidth);

    // Emision of the signal detecting left eye ...
    emit zoneScanned(QString("left eye"));

    superScene->addRect((qreal)px,(qreal)py,(qreal)imWidth,(qreal)imHeight,  *(new QPen( * (new QColor(25
5,0,0)))));
    fprintf(stderr,"# Left eye #\tx: %d\ty: %d\theight: %d\twidth: %d\n",px,py,imHeight,imWidth);

    // Check there is a left eye
    if (px < 1 || py < 1 || imWidth < 1 || imHeight < 1) return 1;

    // Selection of the first boundary for the ears
    midYeyeL = py + imHeight/2;
    extremeEyeL = px + 0.25 * imWidth;
    // Boundary for the nose allocation
    minYeye = eyesY + imHeight;

    // Boundary for the point 12 in the paper (between both eyes)
    maxXeyeL = px + 0.75 * imWidth;

    //Boundary for the left eyebrow up point
    avgXeyeL = (px + 0.25 * imWidth + px + imWidth/2)/2;
    // Move the nodes where we think they should be... important, an offset of 0.25*size is added


        foreach (NodeAutomatic *node, nodes)
        {
            if (nodei == 8) node->setPos(px + 0.25 * imWidth, py + imHeight/2);
            if (nodei == 9) node->setPos(px + 0.75 * imWidth, py + imHeight/2);
            if (nodei == 10) node->setPos(px + imWidth/2, py + 0.25 * imHeight);
            if (nodei == 11) node->setPos(px + imWidth/2, py + 0.75 * imHeight);
            nodei++;
        }


    // ####################################################################################
    // -------- Third: Right eye recognition inside left upper quarter  -------------------
    // ####################################################################################


    subImageEyeR = cvCreateImage(cvSize(result.width(), result.height()), IPL_DEPTH_8U, 3);

    // Clean the image
    for (int i = 0; i < subImageEyeR->width*subImageEyeR->height*3; i++) // over w*h*3 (channels)
        ((char*)subImageEyeR->imageData)[i] = 255;

    // Copy the information between both image formats. Only the face region is needed to locate the eye s
quares
    h = eyesX + eyesW/2;
    for (int i = eyesY ; i < eyesY + eyesH;i++)
    {
        for (int j = 3*(eyesX + eyesW/2); j <  (eyesX + eyesW)*3; j+=3)
        {

            ((char*)subImageEyeR->imageData)[i*result.width()*3 + j  + 0] = qBlue(result.pixel(h,
i));    // blue channel
            ((char*)subImageEyeR->imageData)[i*result.width()*3 + j  + 1] = qGreen(result.pixel(h,
i));  // green channel
            ((char*)subImageEyeR->imageData)[i*result.width()*3 + j  + 2] = qRed(result.pixel(h,i));   //
red channel
            h++;
        }h=eyesX + eyesW/2;
    }
   // cvSaveImage("right.jpg",subImageEyeR);
    dEyeR = new Detector(subImageEyeR);
    dEyeR->DetectAndDraw('R',&px,&py,&imHeight,&imWidth);

    // Emision of the signal detecting right eye ...
    emit zoneScanned(QString("right eye"));

    superScene->addRect((qreal)px,(qreal)py,(qreal)imWidth,(qreal)imHeight,  *(new QPen( * (new QColor(0,
0,255)))));
    fprintf(stderr,"# Right eye #\tx: %d\ty: %d\theight: %d\twidth: %d\n",px,py,imHeight,imWidth);

    // Check there is a right eye
    if (px < 1 || py < 1 || imWidth < 1 || imHeight < 1) return 1;

    // Find the minimum height position between both eyes for the nose allocation
    if  (py + imHeight > minYeye) minYeye = py + imHeight;

    // Selection of the first boundary for the ears
    midYeyeR = py + imHeight/2;
    extremeEyeR = px + 0.75 * imWidth;


    // Boundary
    minXeyeR = px + 0.25 * imWidth;

    //Boundary for the right eyebrow up point
    avgXeyeR = (px + 0.75 * imWidth + px + imWidth/2)/2;

    // Move the nodes where we think they should be... important, an offset of 0.25*size is added
   nodei = 0;
    foreach (NodeAutomatic *node, nodes)
    {

        if (nodei == 13) node->setPos(px + 0.25 * imWidth, py + imHeight/2);
        if (nodei == 14) node->setPos(px + 0.75 * imWidth, py + imHeight/2);
        if (nodei == 15) node->setPos(px + imWidth/2, py + 0.25 * imHeight);
        if (nodei == 16) node->setPos(px + imWidth/2, py + 0.75 * imHeight);

        // Position of the node between both eyes
        if (nodei == 12) node->setPos((px + 0.25 * imWidth + maxXeyeL)/2, py + imHeight/2 );

        // Position of the nodes for the eyebrows

        if (nodei == 3) node->setPos(avgXeyeL, (px + 0.25 * imWidth + maxXeyeL)/2 );
        if (nodei == 4) node->setPos(maxXeyeL, (px + 0.25 * imWidth + maxXeyeL)/2 );
        if (nodei == 6) node->setPos(minXeyeR, (px + 0.25 * imWidth + maxXeyeL)/2 );
        if (nodei == 7) node->setPos(avgXeyeR, (px + 0.25 * imWidth + maxXeyeL)/2 );


    }
    // ###################################################################################
    // -------- Fourth: Mouth recognition in the center side  -------------------
    // ###################################################################################


    subImageMouth = cvCreateImage(cvSize(result.width(), result.height()), IPL_DEPTH_8U, 3);

    // Clean the image
    for (int i = 0; i < subImageMouth->width*subImageMouth->height*3; i++) // over w*h*3 (channels)
        ((char*)subImageMouth->imageData)[i] = 255;

    // Copy the information between both image formats. Only the face region is needed to locate the eye s
quares
    h=faceX;
    for (int i = faceY + faceH*3/4 ; i < faceY + faceH;i++)
    {
        for (int j = 3*faceX; j <  (faceX + faceW)*3; j+=3)
        {

            ((char*)subImageMouth->imageData)[i*result.width()*3 + j  + 0] = qBlue(result.pixel(h,
i));    // blue channel
            ((char*)subImageMouth->imageData)[i*result.width()*3 + j  + 1] = qGreen(result.pixel(h,
i));  // green channel
            ((char*)subImageMouth->imageData)[i*result.width()*3 + j  + 2] = qRed(result.pixel(h,
i));   // red channel
            h++;
        }h=faceX;
    }
    //cvSaveImage("mouth.jpg",subImageMouth);
    dMouth = new Detector(subImageMouth);
    dMouth->DetectAndDraw('M',&px,&py,&imHeight,&imWidth);

    // Emision of the signal detecting mouth ...
    emit zoneScanned(QString("mouth"));

    superScene->addRect((qreal)px,(qreal)py,(qreal)imWidth,(qreal)imHeight,  *(new QPen( * (new QColor(0,2
55,255)))));
    fprintf(stderr,"# Mouth #\tx: %d\ty: %d\theight: %d\twidth: %d\n",px,py,imHeight,imWidth);

    // Check there is a mouth
    if (px < 1 || py < 1 || imWidth < 1 || imHeight < 1) return 1;
    if (px + imWidth > result.width()) return 1;

    mouthY = py;


    // Move the nodes where we think they should be... important, an offset of 0.15*size is added
   nodei = 0;

    foreach (NodeAutomatic *node, nodes)
    {
        if (nodei == 23) node->setPos(px + 0.00 * imWidth, py + imHeight/2);
        if (nodei == 24) node->setPos(px + 1.00 * imWidth, py + imHeight/2);
        if (nodei == 25) node->setPos(px + imWidth/2, py + 0.00 * imHeight);
        if (nodei == 26) node->setPos(px + imWidth/2, py + 1.00 * imHeight);

        nodei++;
    }

    // ###################################################################################
    // -------- Fifth: Nose recognition inside the center side  -------------------
    // ###################################################################################


    subImageNose = cvCreateImage(cvSize(result.width(), result.height()), IPL_DEPTH_8U, 3);

    // Clean the image
    for (int i = 0; i < subImageNose->width*subImageNose->height*3; i++) // over w*h*3 (channels)
        ((char*)subImageNose->imageData)[i] = 255;
    // Copy the information between both image formats. Only the face region is needed to locate the eye s
quares
    h=faceX;
    if (mouthY == 0 ) mouthY = faceY + faceH;
    for (int i = minYeye - eyesH/2 ; i < mouthY;i++)
    {
        for (int j = 3*faceX; j <  (faceX + faceW)*3; j+=3)
        {

            ((char*)subImageNose->imageData)[i*result.width()*3 + j  + 0] = qBlue(result.pixel(h,
i));    // blue channel
            ((char*)subImageNose->imageData)[i*result.width()*3 + j  + 1] = qGreen(result.pixel(h,
i));  // green channel
            ((char*)subImageNose->imageData)[i*result.width()*3 + j  + 2] = qRed(result.pixel(h,i));   //
red channel
            h++;
        }h=faceX;
    }
    //cvSaveImage("nose.jpg",subImageNose);
    dNose = new Detector(subImageNose);
    dNose->DetectAndDraw('N',&px,&py,&imHeight,&imWidth);

    // Emision of the signal detecting nose ...
    emit zoneScanned(QString("nose"));

    superScene->addRect((qreal)px,(qreal)py,(qreal)imWidth,(qreal)imHeight,  *(new QPen( * (new QColor(12
7,0,127)))));
    fprintf(stderr,"# Nose #\tx: %d\ty: %d\theight: %d\twidth: %d\n",px,py,imHeight,imWidth);

    // Check there is a nose
    if (px < 1 || py < 1 || imWidth < 1 || imHeight < 1) return 1;
    if (px + imWidth > result.width()) return 1;

    // Move the nodes where we think they should be... important, an offset of 0.10*size is added
   nodei = 0;
    if ((px != 0) and (py != 0))
    foreach (NodeAutomatic *node, nodes)
    {
        if (nodei == 19) node->setPos(px + 0.10 * imWidth, py + imHeight/2);
        if (nodei == 20) node->setPos(px + 0.90 * imWidth, py + imHeight/2);
        if (nodei == 21) node->setPos(px + imWidth/2, py + 0.10 * imHeight);
        if (nodei == 22) node->setPos(px + imWidth/2, py + 0.90 * imHeight);


        node->calculateForces();
        //if (nodei == 6) node->setPos((faceX+faceW),midYeyeR );
        nodei++;

    }


return 0;


}


long StoneFaceWidget::lineLength(int A1, int B1, int A2, int B2)
{
    long X1 = A2 - A1;
    long X2 = B2 - B1;
    return sqrt( X2 * X2 + X1 * X1);
}

double StoneFaceWidget::areaOfTriangle(int Ax, int Ay, int Bx, int By, int Cx, int Cy)
{

return (float) 0.5 * abs(Ax*By + Ay*Cx + Bx*Cy - By*Cx - Ax*Cy - Ay*Bx);
```

```
}


bool StoneFaceWidget::insideTriangle(int Ax, int Ay, int Bx, int By, int Cx, int Cy, int Px, int Py)
{

    float areaInd,areaTotal; //areas

    areaInd = areaOfTriangle(Px,Py,Ax,Ay,Bx,By) + areaOfTriangle(Px,Py,Bx,By,Cx,Cy) + areaOfTriangle(Px,P
y,Cx,Cy,Ax,Ay);
    areaTotal = areaOfTriangle(Ax,Ay,Bx,By,Cx,Cy);
    if ( ((areaInd - areaTotal)==0.0) && (areaInd != 0.0) and (areaTotal != 0.0) ) return true;
    else return false;

}


int StoneFaceWidget::detectElementsTiny(bool stretch)
{

    IplImage *ImageFace,*subImageFace, *subImageEyeR, *subImageEyeL, *subImageNose, *subImageMouth, *subIm
ageEyes;
    Detector *dFace, *dEyeR, *dEyeL, *dMouth, *dNose, *dEyes;
    int px,py,imHeight,imWidth;
    int h = 0;

    // Variables to obtain the boundary conditions

    int midYeyeL, midYeyeR; // Contains the medium height both eyes to determine the ears
    int faceX,faceY,faceW,faceH;
    int eyesX,eyesY,eyesW,eyesH;
    int mouthY,minYeye,maxXeyeL;
    int minXeyeR,avgXeyeR;
    int nodei = 0; // for list moving
    int extremeEyeR,extremeEyeL,avgXeyeL;
    // Creation of a list with the nodes to be moved

    QList<NodeAutomatic *> nodes;
    foreach (QGraphicsItem *item, scene()->items()) {
        if (NodeAutomatic *node = qgraphicsitem_cast<NodeAutomatic *>(item))
            nodes << node;
    }


    // ####################################################################################
    // -------------------- First: Face recognition  -------------------------------------
    // ####################################################################################


    // Extraction of the face of the person...
    ImageFace = cvCreateImage(cvSize(imf1->width(), imf1->height()), IPL_DEPTH_8U, 3);

    for (int i=0;i < imf1->width() * 3 * imf1->height(); i++)
        ((char*)ImageFace->imageData)[i] = *imf1->bits()+i;

    // Creation of the detector and obtaining of the coordinates for the face
    dFace =  new Detector(ImageFace);
    dFace->DetectAndDraw('F',&px,&py,&imHeight,&imWidth);

    // Emision of the signal detecting face ...
    emit zoneScanned(QString("face"));

    // Now we just want the face scaled with the coordinates changed and work with result

    QImage result = imf1->copy(px,py,imHeight,imWidth);
    //printf("face h = %d, w = %d px %d py %d\n", imHeight, imWidth, px,py);
    //if (imWidth == 0 || imHeight == 0) return 1;
    result = result.scaled(180,180,Qt::IgnoreAspectRatio,Qt::SmoothTransformation);
    imf1 = new QImage(result);
    imf2 = new QImage(result);
    QPixmap *k = new QPixmap(result.width(),result.height());
    face1->setPixmap(k->fromImage(result));

    subImageFace = cvCreateImage(cvSize(result.width(), result.height()), IPL_DEPTH_8U, 3);

    for (int i=0;i < result.width() * 3 * result.height(); i++)
        ((char*)subImageFace->imageData)[i] = *result.bits()+i;

    // Extract the face picture in standard size
    QString imagePath(*widgetFileName);
    imagePath.insert(imagePath.size()-4,QString("_std"));
    fprintf(stderr,"path: %s\n",imagePath.toLatin1().data());
    //QString cara = imagePath.append("-cara.jpg");
    //cvSaveImage(cara.toLatin1().data(),subImageFace);

    // Update the coordinates
    px = 0;py = 0;imHeight=result.height();imWidth=result.width();


    superScene->addRect((qreal)px,(qreal)py,(qreal)imWidth,(qreal)imHeight,  *(new QPen( * (new QColor(0,
0,0)))));
    fprintf(stderr,"# FaceT #\tx: %d\ty: %d\theight: %d\twidth: %d\n",px,py,imHeight,imWidth);

    // Fill the face boundary variables. Throught the face boundaries the rest of the elements will be pro
vided
    faceX = px; faceY = py; faceW = imWidth; faceH = imHeight;

    // Check there is a face recogniced to continue
    if (px == 0 && py == 0 && imWidth == 0 && imHeight == 0) return 1;
    // Move the nodes where we think they should be...

    // ####################################################################################
    // --------            Second: Both eyes recognition               -------------------
    // ####################################################################################

    //cvSaveImage("eyes.jpg",subImageEyes);
    subImageEyes = cvCreateImage(cvSize(result.width(), result.height()), IPL_DEPTH_8U, 3);

    // Clean the image
    for (int i = 0; i < subImageEyes->width*subImageEyes->height*3; i++) // over w*h*3 (channels)
            ((char*)subImageEyes->imageData)[i] = 255;

    // Copy the information between both image formats. Only the face region is needed to locate the eye s
quares
    h=faceX;
    for (int i = faceY ; i < faceY + faceH*3/4;i++)
    {
        for (int j = 3* faceX; j <  (faceX + faceW)*3; j+=3)
        {

            ((char*)subImageEyes->imageData)[i*result.width()*3 + j + 0] = qBlue(result.pixel(h,
i));    // blue channel
            ((char*)subImageEyes->imageData)[i*result.width()*3 + j + 1] = qGreen(result.pixel(h,i));  //
green channel
            ((char*)subImageEyes->imageData)[i*result.width()*3 + j + 2] = qRed(result.pixel(h,i));   // r
ed channel
            h++;
        }h=faceX;
    }


    // Save the image (debugging)
    //cvSaveImage("eyes.jpg",subImageEyes);

    //Creation of the detector object and obtention of the coordinates
    dEyes = new Detector(subImageEyes);
    dEyes->DetectAndDraw('E',&px,&py,&imHeight,&imWidth);

    // Emision of the signal detecting eyes ...
    emit zoneScanned(QString("eyes"));

    superScene->addRect((qreal)px,(qreal)py,(qreal)imWidth,(qreal)imHeight,  *(new QPen( * (new QColor(25
5,0,255)))));
    fprintf(stderr,"# Eyes #\tx: %d\ty: %d\theight: %d\twidth: %d\n",px,py,imHeight,imWidth);

    // Fill the eyes boundary variables. Throught the face boundaries the eyes will be provided
    eyesX = px; eyesY = py; eyesW = imWidth; eyesH = imHeight;

    // Check there are eyes
    if (px < 1 || py < 1 || imWidth < 1 || imHeight < 1) return 1;




    // ##################################################################################
    // -------- Third: Left eye recognition inside left half eyes region  ------------------
    // ##################################################################################


    subImageEyeL = cvCreateImage(cvSize(result.width(), result.height()), IPL_DEPTH_8U, 3);

    // Clean the image
    for (int i = 0; i < subImageEyeL->width*subImageEyeL->height*3; i++) // over w*h*3 (channels)
            ((char*)subImageEyeL->imageData)[i] = 255;

    // Copy the information between both image formats. Only the face region is needed to locate the eye s
quares
    h=eyesX;
    for (int i = eyesY ; i < eyesY + eyesH;i++)
    {
        for (int j = 3* eyesX; j <  (eyesX + eyesW/2)*3; j+=3)
        {

            ((char*)subImageEyeL->imageData)[i*result.width()*3 + j + 0] = qBlue(result.pixel(h,
i));    // blue channel
            ((char*)subImageEyeL->imageData)[i*result.width()*3 + j + 1] = qGreen(result.pixel(h,i));  //
green channel
            ((char*)subImageEyeL->imageData)[i*result.width()*3 + j + 2] = qRed(result.pixel(h,i));   // r
ed channel
            h++;
        }h=eyesX;
    }


    // Save the image (debugging)
    //cvSaveImage("left.jpg",subImageEyeL);

    //Creation of the detector object and obtention of the coordinates
    dEyeL = new Detector(subImageEyeL);
    dEyeL->DetectAndDraw('L',&px,&py,&imHeight,&imWidth);

    minYeye = py + imHeight;

    // Emision of the signal detecting left eye ...
    emit zoneScanned(QString("left eye"));

    superScene->addRect((qreal)px,(qreal)py,(qreal)imWidth,(qreal)imHeight,  *(new QPen( * (new QColor(25
5,0,0)))));
    fprintf(stderr,"# Left eye #\tx: %d\ty: %d\theight: %d\twidth: %d\n",px,py,imHeight,imWidth);

    // Check there is a left eye
    if (px < 1 || py < 1 || imWidth < 1 || imHeight < 1) return 1;

        foreach (NodeAutomatic *node, nodes)
        {
            if (nodei == 3) node->setPos(px + 0.5 * imWidth, py + 0.5 + imHeight);
            nodei++;
        }
    // ####################################################################################
    // -------- Third: Right eye recognition inside left upper quarter  -------------------
    // ####################################################################################


    subImageEyeR = cvCreateImage(cvSize(result.width(), result.height()), IPL_DEPTH_8U, 3);

    // Clean the image
    for (int i = 0; i < subImageEyeR->width*subImageEyeR->height*3; i++) // over w*h*3 (channels)
        ((char*)subImageEyeR->imageData)[i] = 255;

    // Copy the information between both image formats. Only the face region is needed to locate the eye s
quares
    h = eyesX + eyesW/2;
    for (int i = eyesY ; i < eyesY + eyesH;i++)
    {
        for (int j = 3*(eyesX + eyesW/2); j <  (eyesX + eyesW)*3; j+=3)
        {

            ((char*)subImageEyeR->imageData)[i*result.width()*3 + j  + 0] = qBlue(result.pixel(h,
i));    // blue channel
            ((char*)subImageEyeR->imageData)[i*result.width()*3 + j  + 1] = qGreen(result.pixel(h,
i));  // green channel
            ((char*)subImageEyeR->imageData)[i*result.width()*3 + j  + 2] = qRed(result.pixel(h,i));   //
red channel
            h++;
        }h=eyesX + eyesW/2;
    }
   // cvSaveImage("right.jpg",subImageEyeR);
    dEyeR = new Detector(subImageEyeR);
    dEyeR->DetectAndDraw('R',&px,&py,&imHeight,&imWidth);

    // Emision of the signal detecting right eye ...
    emit zoneScanned(QString("right eye"));

    superScene->addRect((qreal)px,(qreal)py,(qreal)imWidth,(qreal)imHeight,  *(new QPen( * (new QColor(0,
0,255)))));
    fprintf(stderr,"# Right eye #\tx: %d\ty: %d\theight: %d\twidth: %d\n",px,py,imHeight,imWidth);

    // Check there is a right eye
    if (px < 1 || py < 1 || imWidth < 1 || imHeight < 1) return 1;

    if (py + imHeight > minYeye ) minYeye = py + imHeight;
    // Move the nodes where we think they should be...
   nodei = 0;
    foreach (NodeAutomatic *node, nodes)
    {
        if (nodei == 4) node->setPos(px + 0.5 * imWidth, py + 0.5 * imHeight);
        nodei++;
    }

    // ####################################################################################
    // -------- Fourth: Mouth recognition in the center side  -------------------
    // ####################################################################################


    subImageMouth = cvCreateImage(cvSize(result.width(), result.height()), IPL_DEPTH_8U, 3);

    // Clean the image
    for (int i = 0; i < subImageMouth->width*subImageMouth->height*3; i++) // over w*h*3 (channels)
        ((char*)subImageMouth->imageData)[i] = 255;

    // Copy the information between both image formats. Only the face region is needed to locate the eye s
quares
    h=faceX;
    for (int i = faceY + faceH*3/4 ; i < faceY + faceH;i++)
    {
        for (int j = 3*faceX; j <  (faceX + faceW)*3; j+=3)
        {
            ((char*)subImageMouth->imageData)[i*result.width()*3 + j  + 0] = qBlue(result.pixel(h,
i));    // blue channel
            ((char*)subImageMouth->imageData)[i*result.width()*3 + j  + 1] = qGreen(result.pixel(h,
i));  // green channel
            ((char*)subImageMouth->imageData)[i*result.width()*3 + j  + 2] = qRed(result.pixel(h,
i));   // red channel
            h++;
        }h=faceX;
    }
    //cvSaveImage("mouth.jpg",subImageMouth);
    dMouth = new Detector(subImageMouth);
    dMouth->DetectAndDraw('M',&px,&py,&imHeight,&imWidth);

    // Emision of the signal detecting mouth ...
    emit zoneScanned(QString("mouth"));

    superScene->addRect((qreal)px,(qreal)py,(qreal)imWidth,(qreal)imHeight,  *(new QPen( * (new QColor(0,2
55,255)))));
    fprintf(stderr,"# Mouth #\tx: %d\ty: %d\theight: %d\twidth: %d\n",px,py,imHeight,imWidth);

    // Check there is a mouth
    if (px < 1 || py < 1 || imWidth < 1 || imHeight < 1) return 1;
    if (px + imWidth > result.width()) return 1;



    // Move the nodes where we think they should be...
   nodei = 0;

    foreach (NodeAutomatic *node, nodes)
    {
        if (nodei == 8) node->setPos(px + 0.5 * imWidth, py + 0.5 * imHeight);
        nodei++;
    }

    // ####################################################################################
    // -------- Fifth: Nose recognition inside the center side  -------------------
    // ####################################################################################


    subImageNose = cvCreateImage(cvSize(result.width(), result.height()), IPL_DEPTH_8U, 3);

    // Clean the image
    for (int i = 0; i < subImageNose->width*subImageNose->height*3; i++) // over w*h*3 (channels)
        ((char*)subImageNose->imageData)[i] = 255;

    // Copy the information between both image formats. Only the face region is needed to locate the eye s
quares
    h=faceX;
    if (mouthY == 0 ) mouthY = faceY + faceH;
    for (int i = minYeye - eyesH/2 ; i < mouthY;i++)
    {
        for (int j = 3*faceX; j <  (faceX + faceW)*3; j+=3)
        {

            ((char*)subImageNose->imageData)[i*result.width()*3 + j  + 0] = qBlue(result.pixel(h,
i));    // blue channel
            ((char*)subImageNose->imageData)[i*result.width()*3 + j  + 1] = qGreen(result.pixel(h,
i));  // green channel
            ((char*)subImageNose->imageData)[i*result.width()*3 + j  + 2] = qRed(result.pixel(h,i));   //
red channel
            h++;
        }h=faceX;
    }
    //cvSaveImage("nose.jpg",subImageNose);
    dNose = new Detector(subImageNose);
    dNose->DetectAndDraw('N',&px,&py,&imHeight,&imWidth);

    // Emision of the signal detecting nose ...
    emit zoneScanned(QString("nose"));

    superScene->addRect((qreal)px,(qreal)py,(qreal)imWidth,(qreal)imHeight,  *(new QPen( * (new QColor(12
7,0,127)))));
    fprintf(stderr,"# Nose #\tx: %d\ty: %d\theight: %d\twidth: %d\n",px,py,imHeight,imWidth);

    // Check there is a nose
    if (px < 1 || py < 1 || imWidth < 1 || imHeight < 1) return 1;
    if (px + imWidth > result.width()) return 1;

    // Move the nodes where we think they should be... important, an offset of 0.10*size is added
   nodei = 0;
    if ((px != 0) and (py != 0))
    foreach (NodeAutomatic *node, nodes)
    {
        if (nodei == 6) node->setPos(px + 0.5 * imWidth, py + 0.5 * imHeight);
        node->calculateForces();
        nodei++;
    }


return 0;


}



// Master Thesis - StoneFace v.1.0 beta C by Jorge Garcia Bueno
// Universidad Carlos III de Madrid + University of Glasgow 2009
// All rights reserved by the author
// ---------------------------------------------------------------
// welcomewidget.h - Header with the instructions for welcomewidget.cpp

#ifndef WELCOMEWIDGET_H
#define WELCOMEWIDGET_H

#include <QtGui>
#include <QPixmap>
#include <QGraphicsView>
#include <QListView>
#include <QDialog>
#include <QPushButton>
#include <QDesktopWidget>
#include "mainwindow.h"


class WelcomeWidget : public QWidget
{
    Q_OBJECT


private:
    // MainWindow handle
    QMainWindow *mainWin;
    void createActions();
    void MoveCenterScreen();


public:
    WelcomeWidget(QMainWindow *parent);
    void mainWindowOrder(int);
    int itemSelected(void);
    // List with the options
    QListWidget *listWidget;
};


#endif
```

```
// Master Thesis - StoneFace v.1.0 beta C by Jorge Garcia Bueno
// Universidad Carlos III de Madrid + University of Glasgow 2009
// All rights reserved by the author
// ----------------------------------------------------------------
// welcomewidget.cpp - Starting interface that controls the option menu

#include "welcomewidget.h"

WelcomeWidget::WelcomeWidget(QMainWindow *parent)
{
    // Remember the handle of the parent for the returning function
    mainWin = parent;

    // Create the selection menu
    listWidget = new QListWidget(this);

    // Defining the interface of the object
    listWidget->setIconSize(QSize(128,128));
    listWidget->resize(QSize(3*128+100,180));
    listWidget->setViewMode(QListView::IconMode);
    listWidget->setMovement(QListView::Static);
    listWidget->setContextMenuPolicy( Qt::CustomContextMenu );

    // Creating the sub-items of the list
    QListWidgetItem *but1 = new QListWidgetItem;
    but1->setText("Create a new manual face");
    but1->setIcon(QIcon("./images/manual.png"));


    QListWidgetItem *but2 = new QListWidgetItem;
    but2->setText("Create a new automatic face");
    but2->setIcon(QIcon("./images/automatic.png"));

    QListWidgetItem *but3 = new QListWidgetItem;
    but3->setText("Search a face");
    but3->setIcon(QIcon("./images/search.png"));

    // Adding them to the list object
    listWidget->addItem(but1);
    listWidget->addItem(but2);
    listWidget->addItem(but3);

    //Start the connections
    createActions();
}
int WelcomeWidget::itemSelected()
{
    if ( listWidget->currentItem()->text() == QString("Create a new manual face") )
        return 0;
    if ( listWidget->currentItem()->text() == QString("Create a new automatic face") )
        return 1;
    if ( listWidget->currentItem()->text() == QString("Search a face") )
        return 2;
    //printf("pressed number %s\n",listWidget->currentItem()->text().toLatin1().data());

}

void WelcomeWidget::createActions()
{
    // Actions related to the buttons clicked
        connect(listWidget, SIGNAL(itemSelectionChanged()), mainWin,SLOT(welcomeButton()) );
}


#############################################################################
# Makefile for building: stoneface
# Generated by qmake (2.01a) (Qt 4.5.0) on: Sat May 16 21:05:58 2009
# Project:  stoneface.pro
# Template: app
# Command: /usr/bin/qmake -unix -o Makefile stoneface.pro
```

```
#############################################################################

####### Compiler, tools and options

CC            = gcc
CXX           = g++
DEFINES       = -DQT_NO_DEBUG -DQT_GUI_LIB -DQT_CORE_LIB -DQT_SHARED
CFLAGS        = -pipe -O2 -Wall -W -D_REENTRANT $(DEFINES)
CXXFLAGS      = -pipe -O2 -Wall -W -D_REENTRANT $(DEFINES)
INCPATH       = -I/usr/share/qt4/mkspecs/linux-g++ -I. -I/usr/include/qt4/QtCore -I/usr/include/qt4/QtGui
-I/usr/include/qt4 -I. -I. -I.
LINK          = g++
LFLAGS        = -Wl,-O1
LIBS          = $(SUBLIBS)  -L/usr/lib -lQtGui -lQtCore -lpthread -lm -lcv -lhighgui -lcvaux
AR            = ar cqs
RANLIB        = 
QMAKE         = /usr/bin/qmake
TAR           = tar -cf
COMPRESS      = gzip -9f
COPY          = cp -f
SED           = sed
COPY_FILE     = $(COPY)
COPY_DIR      = $(COPY) -r
INSTALL_FILE  = install -m 644 -p
INSTALL_DIR   = $(COPY_DIR)
INSTALL_PROGRAM = install -m 755 -p
DEL_FILE      = rm -f
SYMLINK       = ln -sf
DEL_DIR       = rmdir
MOVE          = mv -f
CHK_DIR_EXISTS= test -d
MKDIR         = mkdir -p

####### Output directory

OBJECTS_DIR   = ./

####### Files

SOURCES       = automaticwidget.cpp \
		cameraCapture.cpp \
		detector.cpp \
		eigenfaces.cpp \
		graphwidget.cpp \
		main.cpp \
		mainwindow.cpp \
		mixer.cpp \
		mixeraut.cpp \
		node.cpp \
		nodeaut.cpp \
		searchwidget.cpp \
		stonefacewidget.cpp \
		welcomewidget.cpp moc_automaticwidget.cpp \
		moc_eigenfaces.cpp \
		moc_graphwidget.cpp \
		moc_mainwindow.cpp \
		moc_mixeraut.cpp \
		moc_searchwidget.cpp \
		moc_stonefacewidget.cpp \
		moc_welcomewidget.cpp
OBJECTS       = automaticwidget.o \
		cameraCapture.o \
		detector.o \
		eigenfaces.o \
		graphwidget.o \
		main.o \
		mainwindow.o \
		mixer.o \
		mixeraut.o \
		node.o \
        nodeaut.o \
        searchwidget.o \
        stonefacewidget.o \
        welcomewidget.o \
        moc_automaticwidget.o \
        moc_eigenfaces.o \
        moc_graphwidget.o \
        moc_mainwindow.o \
        moc_mixeraut.o \
        moc_searchwidget.o \
        moc_stonefacewidget.o \
        moc_welcomewidget.o
DIST          = /usr/share/qt4/mkspecs/common/g++.conf \
        /usr/share/qt4/mkspecs/common/unix.conf \
        /usr/share/qt4/mkspecs/common/linux.conf \
        /usr/share/qt4/mkspecs/qconfig.pri \
        /usr/share/qt4/mkspecs/features/qt_functions.prf \
        /usr/share/qt4/mkspecs/features/qt_config.prf \
        /usr/share/qt4/mkspecs/features/exclusive_builds.prf \
        /usr/share/qt4/mkspecs/features/default_pre.prf \
        /usr/share/qt4/mkspecs/features/release.prf \
        /usr/share/qt4/mkspecs/features/default_post.prf \
        /usr/share/qt4/mkspecs/features/warn_on.prf \
        /usr/share/qt4/mkspecs/features/qt.prf \
        /usr/share/qt4/mkspecs/features/unix/thread.prf \
        /usr/share/qt4/mkspecs/features/moc.prf \
        /usr/share/qt4/mkspecs/features/resources.prf \
        /usr/share/qt4/mkspecs/features/uic.prf \
        /usr/share/qt4/mkspecs/features/yacc.prf \
        /usr/share/qt4/mkspecs/features/lex.prf \
        stoneface.pro
QMAKE_TARGET  = stoneface
DESTDIR       = 
TARGET        = stoneface

first: all
####### Implicit rules

.SUFFIXES: .o .c .cpp .cc .cxx .C

.cpp.o:
	$(CXX) -c $(CXXFLAGS) $(INCPATH) -o "$@" "$<"

.cc.o:
	$(CXX) -c $(CXXFLAGS) $(INCPATH) -o "$@" "$<"

.cxx.o:
	$(CXX) -c $(CXXFLAGS) $(INCPATH) -o "$@" "$<"

.C.o:
	$(CXX) -c $(CXXFLAGS) $(INCPATH) -o "$@" "$<"

.c.o:
	$(CC) -c $(CFLAGS) $(INCPATH) -o "$@" "$<"

####### Build rules

all: Makefile $(TARGET)

$(TARGET):  $(OBJECTS)  
	$(LINK) $(LFLAGS) -o $(TARGET) $(OBJECTS) $(OBJCOMP) $(LIBS)

Makefile: stoneface.pro  /usr/share/qt4/mkspecs/linux-g++/qmake.conf /usr/share/qt4/mkspecs/common/g++.con
f \
        /usr/share/qt4/mkspecs/common/unix.conf \
        /usr/share/qt4/mkspecs/common/linux.conf \
        /usr/share/qt4/mkspecs/qconfig.pri \
        /usr/share/qt4/mkspecs/features/qt_functions.prf \
        /usr/share/qt4/mkspecs/features/qt_config.prf \
        /usr/share/qt4/mkspecs/features/exclusive_builds.prf \
        /usr/share/qt4/mkspecs/features/default_pre.prf \
        /usr/share/qt4/mkspecs/features/release.prf \
        /usr/share/qt4/mkspecs/features/default_post.prf \
        /usr/share/qt4/mkspecs/features/warn_on.prf \
        /usr/share/qt4/mkspecs/features/qt.prf \
        /usr/share/qt4/mkspecs/features/unix/thread.prf \
        /usr/share/qt4/mkspecs/features/moc.prf \
        /usr/share/qt4/mkspecs/features/resources.prf \
        /usr/share/qt4/mkspecs/features/uic.prf \
        /usr/share/qt4/mkspecs/features/yacc.prf \
        /usr/share/qt4/mkspecs/features/lex.prf \
        /usr/lib/libQtGui.prl \
        /usr/lib/libQtCore.prl
    $(QMAKE) -unix -o Makefile stoneface.pro
/usr/share/qt4/mkspecs/common/g++.conf:
/usr/share/qt4/mkspecs/common/unix.conf:
/usr/share/qt4/mkspecs/common/linux.conf:
/usr/share/qt4/mkspecs/qconfig.pri:
/usr/share/qt4/mkspecs/features/qt_functions.prf:
/usr/share/qt4/mkspecs/features/qt_config.prf:
/usr/share/qt4/mkspecs/features/exclusive_builds.prf:
/usr/share/qt4/mkspecs/features/default_pre.prf:
/usr/share/qt4/mkspecs/features/release.prf:
/usr/share/qt4/mkspecs/features/default_post.prf:
/usr/share/qt4/mkspecs/features/warn_on.prf:
/usr/share/qt4/mkspecs/features/qt.prf:
/usr/share/qt4/mkspecs/features/unix/thread.prf:
/usr/share/qt4/mkspecs/features/moc.prf:
/usr/share/qt4/mkspecs/features/resources.prf:
/usr/share/qt4/mkspecs/features/uic.prf:
/usr/share/qt4/mkspecs/features/yacc.prf:
/usr/share/qt4/mkspecs/features/lex.prf:
/usr/lib/libQtGui.prl:
/usr/lib/libQtCore.prl:
qmake:  FORCE
    @$(QMAKE) -unix -o Makefile stoneface.pro

dist:
    @$(CHK_DIR_EXISTS) .tmp/stoneface1.0.0 || $(MKDIR) .tmp/stoneface1.0.0
    $(COPY_FILE) --parents $(SOURCES) $(DIST) .tmp/stoneface1.0.0/ && $(COPY_FILE) --parents automaticwidg
et.h cameraCapture.h detector.h eigenfaces.h graphwidget.h mainwindow.h mixer.h mixeraut.h node.h nodeaut.
h searchwidget.h stonefacewidget.h welcomewidget.h /usr/include/opencv/cv.h /usr/include/opencv/cxcore.h /
usr/include/opencv/cxtypes.h /usr/include/opencv/cxerror.h /usr/include/opencv/cvver.h /usr/include/openc
v/cxcore.hpp /usr/include/opencv/cvtypes.h /usr/include/opencv/cv.hpp /usr/include/opencv/cvcompat.h /usr/
include/opencv/highgui.h /usr/include/opencv/cvaux.h /usr/include/opencv/cvaux.hpp /usr/include/opencv/cvv
idsurv.hpp .tmp/stoneface1.0.0/ && $(COPY_FILE) --parents automaticwidget.cpp cameraCapture.cpp detector.c
pp eigenfaces.cpp graphwidget.cpp main.cpp mainwindow.cpp mixer.cpp mixeraut.cpp node.cpp nodeaut.cpp sear
chwidget.cpp stonefacewidget.cpp welcomewidget.cpp .tmp/stoneface1.0.0/ && (cd `dirname .tmp/stoneface1.0.
0` && $(TAR) stoneface1.0.0.tar stoneface1.0.0 && $(COMPRESS) stoneface1.0.0.tar) && $(MOVE) `dirname .tm
p/stoneface1.0.0`/stoneface1.0.0.tar.gz . && $(DEL_FILE) -r .tmp/stoneface1.0.0


clean:compiler_clean
    -$(DEL_FILE) $(OBJECTS)
    -$(DEL_FILE) *~ core *.core


####### Sub-libraries

distclean: clean
    -$(DEL_FILE) $(TARGET)
    -$(DEL_FILE) Makefile


mocclean: compiler_moc_header_clean compiler_moc_source_clean

mocables: compiler_moc_header_make_all compiler_moc_source_make_all

compiler_moc_header_make_all: moc_automaticwidget.cpp moc_eigenfaces.cpp moc_graphwidget.cpp moc_mainwindo
w.cpp moc_mixeraut.cpp moc_searchwidget.cpp moc_stonefacewidget.cpp moc_welcomewidget.cpp
```

```
compiler_moc_header_clean:
	-$(DEL_FILE) moc_automaticwidget.cpp moc_eigenfaces.cpp moc_graphwidget.cpp moc_mainwindow.cpp moc_mix
eraut.cpp moc_searchwidget.cpp moc_stonefacewidget.cpp moc_welcomewidget.cpp
moc_automaticwidget.cpp: mainwindow.h \
		graphwidget.h \
		node.h \
		detector.h \
		/usr/include/opencv/cv.h \
		/usr/include/opencv/cxcore.h \
		/usr/include/opencv/cxtypes.h \
		/usr/include/opencv/cxerror.h \
		/usr/include/opencv/cvver.h \
		/usr/include/opencv/cxcore.hpp \
		/usr/include/opencv/cvtypes.h \
		/usr/include/opencv/cv.hpp \
		/usr/include/opencv/cvcompat.h \
		/usr/include/opencv/highgui.h \
		mixer.h \
		cameraCapture.h \
		eigenfaces.h \
		/usr/include/opencv/cvaux.h \
		/usr/include/opencv/cvaux.hpp \
		/usr/include/opencv/cvvidsurv.hpp \
		searchwidget.h \
		welcomewidget.h \
		automaticwidget.h \
		stonefacewidget.h \
		nodeaut.h \
		mixeraut.h \
		automaticwidget.h
	/usr/bin/moc-qt4 $(DEFINES) $(INCPATH) automaticwidget.h -o moc_automaticwidget.cpp

moc_eigenfaces.cpp: /usr/include/opencv/cv.h \
		/usr/include/opencv/cxcore.h \
		/usr/include/opencv/cxtypes.h \
		/usr/include/opencv/cxerror.h \
		/usr/include/opencv/cvver.h \
		/usr/include/opencv/cxcore.hpp \
		/usr/include/opencv/cvtypes.h \
		/usr/include/opencv/cv.hpp \
		/usr/include/opencv/cvcompat.h \
		/usr/include/opencv/cvaux.h \
		/usr/include/opencv/cvaux.hpp \
		/usr/include/opencv/cvvidsurv.hpp \
		/usr/include/opencv/highgui.h \
		searchwidget.h \
		mainwindow.h \
		graphwidget.h \
		node.h \
		detector.h \
		mixer.h \
		cameraCapture.h \
		eigenfaces.h \
		welcomewidget.h \
		automaticwidget.h \
		stonefacewidget.h \
		nodeaut.h \
		mixeraut.h \
		eigenfaces.h
	/usr/bin/moc-qt4 $(DEFINES) $(INCPATH) eigenfaces.h -o moc_eigenfaces.cpp

moc_graphwidget.cpp: node.h \
		detector.h \
		/usr/include/opencv/cv.h \
		/usr/include/opencv/cxcore.h \
		/usr/include/opencv/cxtypes.h \
		/usr/include/opencv/cxerror.h \
		/usr/include/opencv/cvver.h \
		/usr/include/opencv/cxcore.hpp \
		/usr/include/opencv/cvtypes.h \
        /usr/include/opencv/cv.hpp \
        /usr/include/opencv/cvcompat.h \
        /usr/include/opencv/highgui.h \
        graphwidget.h
    /usr/bin/moc-qt4 $(DEFINES) $(INCPATH) graphwidget.h -o moc_graphwidget.cpp

moc_mainwindow.cpp: graphwidget.h \
        node.h \
        detector.h \
        /usr/include/opencv/cv.h \
        /usr/include/opencv/cxcore.h \
        /usr/include/opencv/cxtypes.h \
        /usr/include/opencv/cxerror.h \
        /usr/include/opencv/cvver.h \
        /usr/include/opencv/cxcore.hpp \
        /usr/include/opencv/cvtypes.h \
        /usr/include/opencv/cv.hpp \
        /usr/include/opencv/cvcompat.h \
        /usr/include/opencv/highgui.h \
        mixer.h \
        cameraCapture.h \
        eigenfaces.h \
        /usr/include/opencv/cvaux.h \
        /usr/include/opencv/cvaux.hpp \
        /usr/include/opencv/cvvidsurv.hpp \
        searchwidget.h \
        mainwindow.h \
        welcomewidget.h \
        automaticwidget.h \
        stonefacewidget.h \
        nodeaut.h \
        mixeraut.h \
        mainwindow.h
    /usr/bin/moc-qt4 $(DEFINES) $(INCPATH) mainwindow.h -o moc_mainwindow.cpp

moc_mixeraut.cpp: stonefacewidget.h \
        nodeaut.h \
        detector.h \
        /usr/include/opencv/cv.h \
        /usr/include/opencv/cxcore.h \
        /usr/include/opencv/cxtypes.h \
        /usr/include/opencv/cxerror.h \
        /usr/include/opencv/cvver.h \
        /usr/include/opencv/cxcore.hpp \
        /usr/include/opencv/cvtypes.h \
        /usr/include/opencv/cv.hpp \
        /usr/include/opencv/cvcompat.h \
        /usr/include/opencv/highgui.h \
        automaticwidget.h \
        mainwindow.h \
        graphwidget.h \
        node.h \
        mixer.h \
        cameraCapture.h \
        eigenfaces.h \
        /usr/include/opencv/cvaux.h \
        /usr/include/opencv/cvaux.hpp \
        /usr/include/opencv/cvvidsurv.hpp \
        searchwidget.h \
        welcomewidget.h \
        mixeraut.h \
        mixeraut.h
    /usr/bin/moc-qt4 $(DEFINES) $(INCPATH) mixeraut.h -o moc_mixeraut.cpp

moc_searchwidget.cpp: mainwindow.h \
        graphwidget.h \
        node.h \
        detector.h \
        /usr/include/opencv/cv.h \
        /usr/include/opencv/cxcore.h \
        /usr/include/opencv/cxtypes.h \
        /usr/include/opencv/cxerror.h \
        /usr/include/opencv/cvver.h \
        /usr/include/opencv/cxcore.hpp \
        /usr/include/opencv/cvtypes.h \
        /usr/include/opencv/cv.hpp \
        /usr/include/opencv/cvcompat.h \
        /usr/include/opencv/highgui.h \
        mixer.h \
        cameraCapture.h \
        eigenfaces.h \
        /usr/include/opencv/cvaux.h \
        /usr/include/opencv/cvaux.hpp \
        /usr/include/opencv/cvvidsurv.hpp \
        searchwidget.h \
        welcomewidget.h \
        automaticwidget.h \
        stonefacewidget.h \
        nodeaut.h \
        mixeraut.h \
        searchwidget.h
    /usr/bin/moc-qt4 $(DEFINES) $(INCPATH) searchwidget.h -o moc_searchwidget.cpp

moc_stonefacewidget.cpp: nodeaut.h \
        stonefacewidget.h \
        detector.h \
        /usr/include/opencv/cv.h \
        /usr/include/opencv/cxcore.h \
        /usr/include/opencv/cxtypes.h \
        /usr/include/opencv/cxerror.h \
        /usr/include/opencv/cvver.h \
        /usr/include/opencv/cxcore.hpp \
        /usr/include/opencv/cvtypes.h \
        /usr/include/opencv/cv.hpp \
        /usr/include/opencv/cvcompat.h \
        /usr/include/opencv/highgui.h \
        stonefacewidget.h
    /usr/bin/moc-qt4 $(DEFINES) $(INCPATH) stonefacewidget.h -o moc_stonefacewidget.cpp

moc_welcomewidget.cpp: mainwindow.h \
        graphwidget.h \
        node.h \
        detector.h \
        /usr/include/opencv/cv.h \
        /usr/include/opencv/cxcore.h \
        /usr/include/opencv/cxtypes.h \
        /usr/include/opencv/cxerror.h \
        /usr/include/opencv/cvver.h \
        /usr/include/opencv/cxcore.hpp \
        /usr/include/opencv/cvtypes.h \
        /usr/include/opencv/cv.hpp \
        /usr/include/opencv/cvcompat.h \
        /usr/include/opencv/highgui.h \
        mixer.h \
        cameraCapture.h \
        eigenfaces.h \
        /usr/include/opencv/cvaux.h \
        /usr/include/opencv/cvaux.hpp \
        /usr/include/opencv/cvvidsurv.hpp \
        searchwidget.h \
        welcomewidget.h \
        automaticwidget.h \
        stonefacewidget.h \
        nodeaut.h \
        mixeraut.h \
        welcomewidget.h
    /usr/bin/moc-qt4 $(DEFINES) $(INCPATH) welcomewidget.h -o moc_welcomewidget.cpp

compiler_rcc_make_all:
compiler_rcc_clean:
```

```
compiler_image_collection_make_all: qmake_image_collection.cpp
compiler_image_collection_clean:
	-$(DEL_FILE) qmake_image_collection.cpp
compiler_moc_source_make_all:
compiler_moc_source_clean:
compiler_uic_make_all:
compiler_uic_clean:
compiler_yacc_decl_make_all:
compiler_yacc_decl_clean:
compiler_yacc_impl_make_all:
compiler_yacc_impl_clean:
compiler_lex_make_all:
compiler_lex_clean:
compiler_clean: compiler_moc_header_clean

####### Compile

automaticwidget.o: automaticwidget.cpp automaticwidget.h \
		mainwindow.h \
		graphwidget.h \
		node.h \
		detector.h \
		/usr/include/opencv/cv.h \
		/usr/include/opencv/cxcore.h \
		/usr/include/opencv/cxtypes.h \
		/usr/include/opencv/cxerror.h \
		/usr/include/opencv/cvver.h \
		/usr/include/opencv/cxcore.hpp \
		/usr/include/opencv/cvtypes.h \
		/usr/include/opencv/cv.hpp \
		/usr/include/opencv/cvcompat.h \
		/usr/include/opencv/highgui.h \
		mixer.h \
		cameraCapture.h \
		eigenfaces.h \
		/usr/include/opencv/cvaux.h \
		/usr/include/opencv/cvaux.hpp \
		/usr/include/opencv/cvvidsurv.hpp \
		searchwidget.h \
		welcomewidget.h \
		stonefacewidget.h \
		nodeaut.h \
		mixeraut.h
	$(CXX) -c $(CXXFLAGS) $(INCPATH) -o automaticwidget.o automaticwidget.cpp

cameraCapture.o: cameraCapture.cpp cameraCapture.h \
		/usr/include/opencv/cv.h \
		/usr/include/opencv/cxcore.h \
		/usr/include/opencv/cxtypes.h \
		/usr/include/opencv/cxerror.h \
		/usr/include/opencv/cvver.h \
		/usr/include/opencv/cxcore.hpp \
		/usr/include/opencv/cvtypes.h \
		/usr/include/opencv/cv.hpp \
		/usr/include/opencv/cvcompat.h \
		/usr/include/opencv/highgui.h \
		eigenfaces.h \
		/usr/include/opencv/cvaux.h \
		/usr/include/opencv/cvaux.hpp \
		/usr/include/opencv/cvvidsurv.hpp \
		searchwidget.h \
		mainwindow.h \
		graphwidget.h \
		node.h \
		detector.h \
		mixer.h \
		welcomewidget.h \
		automaticwidget.h \
		stonefacewidget.h \
		nodeaut.h \
        mixeraut.h
	$(CXX) -c $(CXXFLAGS) $(INCPATH) -o cameraCapture.o cameraCapture.cpp

detector.o: detector.cpp detector.h \
		/usr/include/opencv/cv.h \
		/usr/include/opencv/cxcore.h \
		/usr/include/opencv/cxtypes.h \
		/usr/include/opencv/cxerror.h \
		/usr/include/opencv/cvver.h \
		/usr/include/opencv/cxcore.hpp \
		/usr/include/opencv/cvtypes.h \
		/usr/include/opencv/cv.hpp \
		/usr/include/opencv/cvcompat.h \
		/usr/include/opencv/highgui.h
	$(CXX) -c $(CXXFLAGS) $(INCPATH) -o detector.o detector.cpp

eigenfaces.o: eigenfaces.cpp eigenfaces.h \
		/usr/include/opencv/cv.h \
		/usr/include/opencv/cxcore.h \
		/usr/include/opencv/cxtypes.h \
		/usr/include/opencv/cxerror.h \
		/usr/include/opencv/cvver.h \
		/usr/include/opencv/cxcore.hpp \
		/usr/include/opencv/cvtypes.h \
		/usr/include/opencv/cv.hpp \
		/usr/include/opencv/cvcompat.h \
		/usr/include/opencv/cvaux.h \
		/usr/include/opencv/cvaux.hpp \
		/usr/include/opencv/cvvidsurv.hpp \
		/usr/include/opencv/highgui.h \
		searchwidget.h \
		mainwindow.h \
		graphwidget.h \
		node.h \
		detector.h \
		mixer.h \
		cameraCapture.h \
		welcomewidget.h \
		automaticwidget.h \
		stonefacewidget.h \
		nodeaut.h \
		mixeraut.h
	$(CXX) -c $(CXXFLAGS) $(INCPATH) -o eigenfaces.o eigenfaces.cpp

graphwidget.o: graphwidget.cpp graphwidget.h \
		node.h \
		detector.h \
		/usr/include/opencv/cv.h \
		/usr/include/opencv/cxcore.h \
		/usr/include/opencv/cxtypes.h \
		/usr/include/opencv/cxerror.h \
		/usr/include/opencv/cvver.h \
		/usr/include/opencv/cxcore.hpp \
		/usr/include/opencv/cvtypes.h \
		/usr/include/opencv/cv.hpp \
		/usr/include/opencv/cvcompat.h \
		/usr/include/opencv/highgui.h
	$(CXX) -c $(CXXFLAGS) $(INCPATH) -o graphwidget.o graphwidget.cpp

main.o: main.cpp mainwindow.h \
		graphwidget.h \
		node.h \
		detector.h \
		/usr/include/opencv/cv.h \
		/usr/include/opencv/cxcore.h \
		/usr/include/opencv/cxtypes.h \
		/usr/include/opencv/cxerror.h \
		/usr/include/opencv/cvver.h \
		/usr/include/opencv/cxcore.hpp \
		/usr/include/opencv/cvtypes.h \
	/usr/include/opencv/cv.hpp \
	/usr/include/opencv/cvcompat.h \
	/usr/include/opencv/highgui.h \
	mixer.h \
	cameraCapture.h \
	eigenfaces.h \
	/usr/include/opencv/cvaux.h \
	/usr/include/opencv/cvaux.hpp \
	/usr/include/opencv/cvvidsurv.hpp \
	searchwidget.h \
	welcomewidget.h \
	automaticwidget.h \
	stonefacewidget.h \
	nodeaut.h \
	mixeraut.h
    $(CXX) -c $(CXXFLAGS) $(INCPATH) -o main.o main.cpp

mainwindow.o: mainwindow.cpp mainwindow.h \
	graphwidget.h \
	node.h \
	detector.h \
	/usr/include/opencv/cv.h \
	/usr/include/opencv/cxcore.h \
	/usr/include/opencv/cxtypes.h \
	/usr/include/opencv/cxerror.h \
	/usr/include/opencv/cvver.h \
	/usr/include/opencv/cxcore.hpp \
	/usr/include/opencv/cvtypes.h \
	/usr/include/opencv/cv.hpp \
	/usr/include/opencv/cvcompat.h \
	/usr/include/opencv/highgui.h \
	mixer.h \
	cameraCapture.h \
	eigenfaces.h \
	/usr/include/opencv/cvaux.h \
	/usr/include/opencv/cvaux.hpp \
	/usr/include/opencv/cvvidsurv.hpp \
	searchwidget.h \
	welcomewidget.h \
	automaticwidget.h \
	stonefacewidget.h \
	nodeaut.h \
	mixeraut.h
    $(CXX) -c $(CXXFLAGS) $(INCPATH) -o mainwindow.o mainwindow.cpp

mixer.o: mixer.cpp mixer.h \
	graphwidget.h \
	node.h \
	detector.h \
	/usr/include/opencv/cv.h \
	/usr/include/opencv/cxcore.h \
	/usr/include/opencv/cxtypes.h \
	/usr/include/opencv/cxerror.h \
	/usr/include/opencv/cvver.h \
	/usr/include/opencv/cxcore.hpp \
	/usr/include/opencv/cvtypes.h \
	/usr/include/opencv/cv.hpp \
	/usr/include/opencv/cvcompat.h \
	/usr/include/opencv/highgui.h
    $(CXX) -c $(CXXFLAGS) $(INCPATH) -o mixer.o mixer.cpp

mixeraut.o: mixeraut.cpp mixeraut.h \
	stonefacewidget.h \
	nodeaut.h \
	detector.h \
	/usr/include/opencv/cv.h \
	/usr/include/opencv/cxcore.h \
	/usr/include/opencv/cxtypes.h \
	/usr/include/opencv/cxerror.h \
	/usr/include/opencv/cvver.h \
        /usr/include/opencv/cxcore.hpp \
        /usr/include/opencv/cvtypes.h \
        /usr/include/opencv/cv.hpp \
        /usr/include/opencv/cvcompat.h \
        /usr/include/opencv/highgui.h \
        automaticwidget.h \
        mainwindow.h \
        graphwidget.h \
        node.h \
        mixer.h \
        cameraCapture.h \
        eigenfaces.h \
        /usr/include/opencv/cvaux.h \
        /usr/include/opencv/cvaux.hpp \
        /usr/include/opencv/cvvidsurv.hpp \
        searchwidget.h \
        welcomewidget.h
    $(CXX) -c $(CXXFLAGS) $(INCPATH) -o mixeraut.o mixeraut.cpp

node.o: node.cpp node.h \
        graphwidget.h \
        detector.h \
        /usr/include/opencv/cv.h \
        /usr/include/opencv/cxcore.h \
        /usr/include/opencv/cxtypes.h \
        /usr/include/opencv/cxerror.h \
        /usr/include/opencv/cvver.h \
        /usr/include/opencv/cxcore.hpp \
        /usr/include/opencv/cvtypes.h \
        /usr/include/opencv/cv.hpp \
        /usr/include/opencv/cvcompat.h \
        /usr/include/opencv/highgui.h
    $(CXX) -c $(CXXFLAGS) $(INCPATH) -o node.o node.cpp

nodeaut.o: nodeaut.cpp nodeaut.h \
        stonefacewidget.h \
        detector.h \
        /usr/include/opencv/cv.h \
        /usr/include/opencv/cxcore.h \
        /usr/include/opencv/cxtypes.h \
        /usr/include/opencv/cxerror.h \
        /usr/include/opencv/cvver.h \
        /usr/include/opencv/cxcore.hpp \
        /usr/include/opencv/cvtypes.h \
        /usr/include/opencv/cv.hpp \
        /usr/include/opencv/cvcompat.h \
        /usr/include/opencv/highgui.h
    $(CXX) -c $(CXXFLAGS) $(INCPATH) -o nodeaut.o nodeaut.cpp

searchwidget.o: searchwidget.cpp searchwidget.h \
        mainwindow.h \
        graphwidget.h \
        node.h \
        detector.h \
        /usr/include/opencv/cv.h \
        /usr/include/opencv/cxcore.h \
        /usr/include/opencv/cxtypes.h \
        /usr/include/opencv/cxerror.h \
        /usr/include/opencv/cvver.h \
        /usr/include/opencv/cxcore.hpp \
        /usr/include/opencv/cvtypes.h \
        /usr/include/opencv/cv.hpp \
        /usr/include/opencv/cvcompat.h \
        /usr/include/opencv/highgui.h \
        mixer.h \
        cameraCapture.h \
        eigenfaces.h \
        /usr/include/opencv/cvaux.h \
        /usr/include/opencv/cvaux.hpp \
        /usr/include/opencv/cvvidsurv.hpp \
        welcomewidget.h \
        automaticwidget.h \
        stonefacewidget.h \
        nodeaut.h \
        mixeraut.h
    $(CXX) -c $(CXXFLAGS) $(INCPATH) -o searchwidget.o searchwidget.cpp

stonefacewidget.o: stonefacewidget.cpp stonefacewidget.h \
        nodeaut.h \
        detector.h \
        /usr/include/opencv/cv.h \
        /usr/include/opencv/cxcore.h \
        /usr/include/opencv/cxtypes.h \
        /usr/include/opencv/cxerror.h \
        /usr/include/opencv/cvver.h \
        /usr/include/opencv/cxcore.hpp \
        /usr/include/opencv/cvtypes.h \
        /usr/include/opencv/cv.hpp \
        /usr/include/opencv/cvcompat.h \
        /usr/include/opencv/highgui.h
    $(CXX) -c $(CXXFLAGS) $(INCPATH) -o stonefacewidget.o stonefacewidget.cpp

welcomewidget.o: welcomewidget.cpp welcomewidget.h \
        mainwindow.h \
        graphwidget.h \
        node.h \
        detector.h \
        /usr/include/opencv/cv.h \
        /usr/include/opencv/cxcore.h \
        /usr/include/opencv/cxtypes.h \
        /usr/include/opencv/cxerror.h \
        /usr/include/opencv/cvver.h \
        /usr/include/opencv/cxcore.hpp \
        /usr/include/opencv/cvtypes.h \
        /usr/include/opencv/cv.hpp \
        /usr/include/opencv/cvcompat.h \
        /usr/include/opencv/highgui.h \
        mixer.h \
        cameraCapture.h \
        eigenfaces.h \
        /usr/include/opencv/cvaux.h \
        /usr/include/opencv/cvaux.hpp \
        /usr/include/opencv/cvvidsurv.hpp \
        searchwidget.h \
        automaticwidget.h \
        stonefacewidget.h \
        nodeaut.h \
        mixeraut.h
    $(CXX) -c $(CXXFLAGS) $(INCPATH) -o welcomewidget.o welcomewidget.cpp

moc_automaticwidget.o: moc_automaticwidget.cpp
    $(CXX) -c $(CXXFLAGS) $(INCPATH) -o moc_automaticwidget.o moc_automaticwidget.cpp

moc_eigenfaces.o: moc_eigenfaces.cpp
    $(CXX) -c $(CXXFLAGS) $(INCPATH) -o moc_eigenfaces.o moc_eigenfaces.cpp

moc_graphwidget.o: moc_graphwidget.cpp
    $(CXX) -c $(CXXFLAGS) $(INCPATH) -o moc_graphwidget.o moc_graphwidget.cpp

moc_mainwindow.o: moc_mainwindow.cpp
    $(CXX) -c $(CXXFLAGS) $(INCPATH) -o moc_mainwindow.o moc_mainwindow.cpp

moc_mixeraut.o: moc_mixeraut.cpp
    $(CXX) -c $(CXXFLAGS) $(INCPATH) -o moc_mixeraut.o moc_mixeraut.cpp

moc_searchwidget.o: moc_searchwidget.cpp
    $(CXX) -c $(CXXFLAGS) $(INCPATH) -o moc_searchwidget.o moc_searchwidget.cpp

moc_stonefacewidget.o: moc_stonefacewidget.cpp
    $(CXX) -c $(CXXFLAGS) $(INCPATH) -o moc_stonefacewidget.o moc_stonefacewidget.cpp
```

```
moc_welcomewidget.o: moc_welcomewidget.cpp
	$(CXX) -c $(CXXFLAGS) $(INCPATH) -o moc_welcomewidget.o moc_welcomewidget.cpp

####### Install

install:   FORCE

uninstall:   FORCE

FORCE:
```